ABSTRACT

DATAR, MOHIT ANAND. Priority-based Control Algorithm for Movement of Packages in a Public Distribution Center. (Under the direction of Dr. Michael G. Kay.)

A priority-based control algorithm is presented in this thesis as a control system for the movement of packages in a public distribution center (PDC). The algorithm works under assumptions of the design and automation required in a PDC, as proposed by Kay and seeks to provide a low level yet robust mechanism for guiding the movement of packages. The layout of a PDC is reminiscent of the famous sliding block puzzles, and the packages in the PDC move in a similar fashion to the blocks of the puzzle. An important deviation from the sliding block puzzle is that the algorithm is capable of controlling motion of more than one package at a time. In addition, low-level conflict resolution has been included as a part of the algorithm to ensure that blockages caused due to priority levels and movement sequences are resolved during run time and that no higher level of control is required for resolving the conflict situations. The proposed algorithm is greedy and non-optimal in nature in that it only looks one-step in the future and does not factor in motion planning over multiple time steps. This approach helps to put in a place a simple control logic that is robust and can be scaled for larger operations. A modular approach to the implementation allows for flexibility and increased scope for future enhancements. Algorithm performance is shown by calculating the retrieval times for two scenarios, and it is observed that there is a non-linear increase in the average retrieval time when the warehouse density increases from 90% to almost 100%.
Priority-based Control Algorithm for Movement of Packages in a Public Distribution Center

by
Mohit Anand Datar

A thesis submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the degree of Master of Science

Industrial Engineering

Raleigh, North Carolina
2011

APPROVED BY:

_______________________________  ______________________________
Donald Warsing                Steven Jackson

______________________________
Michael G. Kay
Chair of Advisory Committee
DEDICATION

To Mamma, Baba and Alok
BIOGRAPHY

Mohit Anand Datar was born in December 1985 in Indore, India to Maneesha and Anand Datar. He completed his Bachelor’s in Production Engineering from Bharati Vidyapeeth University, Pune in 2007. Thereafter he worked as a software engineer with Infosys Technologies Limited for a year. In 2008, he joined the family logistics and warehousing business and realized his interest in the field of Supply Chain Management. One year later, in 2009, he joined the Master of Science program in the Edward P. Fitts Department of Industrial and Systems Engineering at North Carolina State University, Raleigh. During his Master’s program, he worked as the supply chain engineering intern with Ryder Integrated Logistics, Alpharetta (GA).
ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Michael Kay for his support, guidance and encouragement throughout the duration of my research. I would also like to thank Dr. Donald Warsing and Dr. Steven Jackson for their guidance and for their acceptance to be on my committee.

I would also like to thank all my friends, especially Aseem Deodhar, Anup Mokashi, Pranav Karambelkar, Saurabh Bakshi, P S Sriram, Avineet Ponde, Priyesh Malegaonkar and Rohan Hangekar for making my time here in Raleigh an extremely enjoyable experience.

Last but not the least; I remain indebted to my parents and my brother for always believing in me and supporting me through all the trouble that i keep getting myself into.
# TABLE OF CONTENTS

**LIST OF TABLES** ............................................................................................................................................... vii

**LIST OF FIGURES** ......................................................................................................................................... viii

**CHAPTER 1: Introduction** ............................................................................................................................. 1

1.1 Introduction ................................................................................................................................................ 1

1.2 Automated Warehousing ............................................................................................................................ 1

1.3 Automated Storage and Retrieval Systems ................................................................................................. 2

1.4 Public Distribution Center .......................................................................................................................... 3

1.5 Research Objectives ................................................................................................................................... 5

1.6 Thesis Outline ........................................................................................................................................... 5

**CHAPTER 2: Literature Review** ................................................................................................................... 7

2.1 Introduction ................................................................................................................................................ 7

2.2 The 15-Puzzle Problem .............................................................................................................................. 7

2.3 The Rush Hour™ Problem .......................................................................................................................... 9

2.4 The Warehouseman Problem ..................................................................................................................... 10

**CHAPTER 3: Control Concept** ................................................................................................................... 13

3.1 Introduction ................................................................................................................................................. 13

3.2 The Layout .................................................................................................................................................. 13

3.3 Requirements and Assumptions ................................................................................................................ 15
3.4 The Control Concept ................................................................. 15

3.5 Simultaneous Movement of Multiple Packages: Conflicts & Blockages .......... 27

CHAPTER 4: Priority-based Control Algorithm ........................................ 33

4.1 Introduction ............................................................................... 33

4.2 The Control Algorithm ............................................................. 33

4.3 Scenario I .................................................................................. 40

4.4 Scenario I - Testing ................................................................... 47

4.5 Scenario II .................................................................................. 48

4.6 Scenario II - Testing ................................................................. 57

CHAPTER 5: Conclusions and Future Work ............................................. 58

REFERENCES .................................................................................. 61

APPENDICES .................................................................................. 63

Appendix A: Scenario I – Algorithm Performance ....................................... 64

Appendix B: Scenario II – Algorithm Performance ..................................... 68

Appendix C: Java code ........................................................................ 72
## LIST OF TABLES

Table 1: Attributes of a module ........................................................................................................ 17
Table 2: Movement of loads over multiple time steps - example................................................. 25
Table 3: Scenario I - Warehouse utilization level - 100% ............................................................. 64
Table 4: Scenario I - Warehouse utilization level - 90% ............................................................... 65
Table 5: Scenario I - Warehouse utilization level - 80% ............................................................... 66
Table 6: Scenario I - Warehouse utilization level - 70% ............................................................... 67
Table 7: Scenario II - Warehouse utilization level - 100% ........................................................... 68
Table 8: Scenario II - Warehouse utilization level - 90% ............................................................. 69
Table 9: Scenario II - Warehouse utilization level - 80% ............................................................. 70
Table 10: Scenario II - Warehouse utilization level - 70% ........................................................... 71
LIST OF FIGURES

Figure 1: Representation of DC layout ............................................................ 14
Figure 2: Overview of the control logic ............................................................... 18
Figure 3: Identification process for modules ..................................................... 19
Figure 4: Illustration of the next location search ............................................. 21
Figure 5: Tagging modules for movement ......................................................... 22
Figure 6: Illustration of the Tagging Modules for Movement process ............... 23
Figure 7: Movement of loads ........................................................................... 25
Figure 8: Illustration of load movement ............................................................ 26
Figure 9: Multiple loads accessing the same resource ....................................... 28
Figure 10: Higher priority order load tagging a lower priority order load .......... 29
Figure 11: Lower priority order load blocked while getting to destination .......... 30
Figure 12: Assignment of a temporary destination to blocked order load .......... 31
Figure 13: Pseudo code for algorithm ............................................................... 34
Figure 14: Legend for the flowcharts ............................................................... 37
Figure 15: Searching for Next Location - Sub process 1 ................................... 38
Figure 16: Tagging modules for movement - Sub Process II ............................ 39
Figure 17: Scenario I for priority based control algorithm ............................... 40
Figure 18: Scenario I - time step t = 0 and time step t = 1 ............................. 41
Figure 19: Scenario I - time step t = 2 and time step t = 3 ............................. 42
Figure 20: Scenario I - time step t = 4 and time step t = 5 ............................. 43
Figure 21: Scenario I - time step t = 6 and time step t = 7 ........................................ 44
Figure 22: Scenario I - time step t = 8 and time step t = 9 ........................................ 45
Figure 23: Scenario I - time step t = 10 and final configuration .................................... 46
Figure 24: Performance of algorithm for scenario I ......................................................... 47
Figure 25: Scenario II for priority based control algorithm ........................................... 48
Figure 26: Scenario II - time step t = 0 and time step t = 1 ........................................... 49
Figure 27: Scenario II - time step t = 2 and time step t = 3 ........................................... 50
Figure 28: Scenario II - time step t = 4 and time step t = 5 ........................................... 51
Figure 29: Scenario II - time step t = 6 and time step t = 7 ........................................... 52
Figure 30: Scenario II - time step t = 8 and time step t = 9 ........................................... 53
Figure 31: Scenario II - time step t = 10 and time step t = 11 ....................................... 54
Figure 32: Scenario II - time step t = 11 and time step t = 12 ....................................... 55
Figure 33: Scenario II - final configuration ....................................................................... 56
Figure 34: Performance of Algorithm for Scenario II ..................................................... 57
CHAPTER 1: Introduction

1.1 Introduction

As organizations look toward reducing operating costs in order to increase profitability, significant attention is given to cutting costs in the supply chain. Distribution centers (DCs) are an important component of a supply chain. In terms of cost; they represent approximately 20 percent of the total logistics costs [1], whilst in terms of service they are critical to the achievement of customer service levels [2]. Automation in DCs is regarded as a key for organizations to increase productivity while achieving significant reduction in long term costs. As the cost of labor and land increases, it has become imperative for organizations to automate their warehouses and maximize the utilization of resources like warehouse space, equipment and labor.

1.2 Automated Warehousing

Warehouse automation has been defined as “The direct control of handling equipment producing movement and storage of loads without the need for operators or drivers” [3]. Automation technologies are used to perform a variety of warehouse operations, for example picking, packing, storing, retrieving, slotting, sorting etc. It is common for an organization to have a mix of manual and automated systems, automating only the most critical of its required operations. Using automation in warehouses helps reduce operating costs by
increasing efficiency, increasing customer satisfaction, increasing accuracy, and improving operational efficiency by maximizing utilization of warehouse space.

In a study conducted by Aberdeen Group in 2007, it was found that warehousing executives were faced with a myriad of market pressures that needed to be addressed in the immediate future to ensure that their operation met customer expectations [4]. Some of the issues cited in the study were insufficient warehouse space, need to satisfy customer orders faster, flexibility to handle increased volumes, high cost and low availability of labor. Automation strategies like Carousels, Automated Storage Retrieval Systems, Automated Guided Vehicles and Robots are some of the solutions that organizations turn to for addressing these issues.

1.3 Automated Storage and Retrieval Systems

An AS/RS system is defined as a storage system that uses fixed-path storage and retrieval machines running on one or more rails between fixed arrays of storage racks. AS/RS is typically used for handling unit loads like pallets and larger units which move as a whole. Loads are moved with the help of the retrieval machines to and from the picking and replenishment areas. For order fulfillment that demands breaking the pallet loads, the AS/RS delivers the unit load (pallet) to the picking area. Manual labor is then used to break the pallet, pick the order and then the remaining items are re-palletized for storage using the automated system.
The most common AS/RS have an aisle-captive crane, which means that a crane is fixed for serving a particular aisle. One variation of the AS/RS uses a single crane to traverse multiple aisles and retrieve loads. AS/RS consists of two parts, one is the automated mechanism that moves around, retrieving and storing loads and the second are the storage racks and shelves from which loads are retrieved and to which they are stored. The racks are standard single-deep or double-deep pallet racks and are serviced by the moving cranes in the aisles. A recent variation of the AS/RS has a lift that moves vertically and deposits the load at the desired level in the rack, and then a horizontal moving guided vehicle picks the unit load and places it in the appropriate location in the rack. Complex designs and multiple design considerations along with high capital and maintenance costs prevent AS/RS from being used across all industries and for all products. The design considerations for AS/RS include a number of aisles, height of racks, length of aisles, cranes per aisle, storage assignment methods, control operations (single/dual command) and a scheduling approach. These make the design complex and difficult to modify once it is set up. Performance of an AS/RS is measured by travel time per request, waiting time for cranes, waiting time of products, and number of pending requests.

1.4 Public Distribution Center

A public logistics network (PLN) is proposed as an alternative to private logistics networks for the ground transport of parcels [5]. The network is a collection of small public distribution centers (PDCs) operating independently. Since there is no centralized control to
the operation, it is important that information be transferred between the independent carriers and the distribution centers. Operation of the PLN is similar to the packages being transported in a network or the Internet. Once a package enters the network, it is then routed through a sequence of DCs located throughout the metropolitan area. The DCs function in a manner similar to routers in a network and route packages not always to the destination, but to the next closest destination in the path to the final destination.

PLN consists of highly automated multi-company distribution centers that can help reduce shipping and handling costs for ground transport of parcels. Since each DC handles a large volume of parcels travelling through the network, economies of scale can be achieved even for a small DC. Kay [6] proposes that in order to be cost effective, all loading, unloading, and sorting activities in the DC must be highly automated, and for that, the primary material handling device in the DCs is a square module with orthogonal pop-up powered wheels. Larger loads are set up on more than one of these modules and thus the smallest material handling device may not be designed for handling the heaviest loads. This design helps the modules to be manufactured cheaply and makes the system cost effective.

The proposed design of the material handling device provides considerable advantages over the current automated systems used in DCs like AS/RS and conveyor systems. It enables high cube utilization, access to any load anywhere in the DC, and low cost implementation. Also, unlike current material handling systems, the smallest material handling device need not be designed for handling the heaviest load, as multiple unit devices can carry a large load,
thereby making the unit devices cheaper to manufacture. The proposed structure aims at making transport of a parcel cheaper or as cheap as shipping an entire truck load.

1.5 Research Objectives

The main objectives of this research are to do the following:

1. Develop a basic algorithm for movement of loads in the PDC. Each package in the DC is assigned a level of priority based on its delivery schedule and the availability of resources for its delivery to the destination. The algorithm uses the unique priority of each load to determine movement within the warehouse.

2. Demonstrate a robust and generic Java-based implementation of the module movement algorithm and analyze various scenarios and conflicts that may arise during module movement in the DC.

3. Document the shortcomings of the proposed algorithm and identify the scope for improvement along with identification of areas where sophisticated controls can be added in the current algorithm.

1.6 Thesis Outline

Chapter 2 of thesis discusses the related literature on this topic. The literature review covers previous work on puzzle-based problems and their solutions along with a puzzle-based storage system. In Chapter 3, the control concept of the proposed algorithm for the movement of packages in a public distribution center is presented, and some issues that arise
during the movement process are discussed. The complete algorithm is presented in Chapter 4 along with its application to a scenarios and a summary of results. Finally Chapter 5 discusses the scope for future work and conclusions of the present work.
CHAPTER 2: Literature Review

2.1 Introduction

The primary objective of this thesis is to propose an algorithm for moving packages in an automated warehouse and thus is closely related to motion planning strategies and algorithms. This chapter looks at related literature on the 15 puzzle, Rush Hour™ Problem and the Warehouseman problem since all three problems are similar in design and construction to the problem at hand and provide an insight to the complexity of moving multiple objects in a constrained space.

2.2 The 15-Puzzle Problem

The famous 15-Puzzle is similar in design and construction to the problem concerned in this research. It consists of a frame of numbered square tiles in random order with one tile missing. The objective of the puzzle is to arrange the tiles in order by moving tiles in the only empty space. Like the problem discussed in this thesis, the tiles are allowed only vertical and horizontal movement. It is of particular interest to this research as at high utilization, the proposed layout would operate similar to a sliding block puzzle. The difference however lies in the fact that the proposed algorithm in this thesis deals with retrieval of objects from any given configuration and density of packages in the layout as against achieving a final
rearranged configuration of packages, as is the case in the 15-Puzzle. This makes the problem in this thesis easier to deal with as compared to the classic 15-Puzzle.

Bauer [7] proposes a combination of Manhattan Distance function and a Pair Distance Heuristic along with the Iterative Deepening A* search algorithm to achieve an 80% reduction in node count in the heuristic search. However a complete solution is not obtained using the proposed heuristic.

Gue and Kim [8] present an optimal algorithm for a puzzle-based storage system. They consider the classic 15 puzzle problem in a warehouse layout. The algorithm has been adapted for more than one empty location, called “escorts” in the proposed algorithm, and the authors propose a heuristic for scenarios with larger layout sizes than the 15 puzzle. The proposed algorithm focuses on retrieval of only one item at a time and makes use of one escort to execute with a minimum retrieval time. In the case of multiple escorts, the proposed algorithm uses a heuristic to choose from the available escorts at each time step. The authors also compare the performance of the proposed system against a traditional retrieval system operating at a high density and report the proposed system to be operating with optimality at a high layout utilization. Optimality is established by calculating the minimum number of steps required to retrieve an item to a location by using the proposed strategy and comparing it to traditional aisle-based retrieval strategies. This thesis introduces unique priority levels for each package, allowing the ability to control movement of more than one object at the same time, as against the work by Gue et al., which defines no priorities specifically but
functions as if there were just two levels of priorities wherein the object to be moved has a higher priority and all other objects have equal lower priority.

2.3 The Rush Hour™ Problem

Rush Hour™ is a grid-based game that involves moving a target car to an exit located on the center of any one edge of the grid through a series of vertical or horizontal steps. The other cars in the grid are moved in such a way that the target car gets a clear path to the exit. Flake and Baum [9] use a lazy form of dual-rail reversible logic such that the movement of “output” cars can occur only if logical combinations of “input” cars can also move. They also establish that this kind of problem and its generalization lies in the PSPACE Complete space, which are among the hardest problems in the PSPACE. They also propose a Generalized Rush Hour (GRH) which is a variant of the original problem with arbitrary width and height of the grid and with the option of locating the exit anywhere on the perimeter of the grid. It has also been demonstrated that the GRH or a variation of it can be realized as a physical system and used for practical motion-planning systems. The GRH takes into account different sizes of the cars and allows only fixed-object orientation. No rotation is permitted for the cars. In the Rush Hour™ problem, the objective is to clear a path for the selected object to move out to its destination whereas the problem in this thesis deals with moving the selected object one step at a time to its destination. The approach in this thesis is different from the Rush Hour™ problem as the proposed algorithm is able to achieve movement of selected objects with less available empty space. Also in a Rush Hour™ problem, the objects
are permitted only vertical or horizontal motion unlike the assumption of this thesis, in which loads can move both vertically and horizontally, thus making the problem at hand easier.

Hearn and Demaine [10] propose the Nondeterministic Constraint Logic (NCL) model of computation by giving simple reductions to show that the classic unrestricted sliding block puzzles are PSPACE-Hard. They also prove that sliding 1×2 blocks (dominoes) around in a box is PSPACE-hard.

2.4 The Warehouseman Problem

The Warehouseman problem [11] involves the coordinated motion-planning of multiple objects confined in a space. The objective is to attain a final configuration of objects by moving objects in the warehouse.

PSPACE hardness of the “warehouseman’s problem” was shown by Hopcroft, Shwartz and Sharir [11]. The research points out that even restricted two-dimensional problem for movement of rectangular objects in a rectangular area are PSPACE hard.

Sharma and Aloimonos [12] introduce a concept called the Temporary Storage Space (TSS), as a general way of constraining free space and use it to propose three algorithms valid under different sets of constraints. The constraints are on the sizes and orientation of the objects/blocks to be arranged. Adequate TSS is proposed for all the cases, to ensure
rearrangement of \( n \) blocks through algorithms having \( O(n^2) \) running time. The algorithms are proposed for square blocks in an \( N \times N \) square layout. The approach to the problem is to find constraints that when imposed on provably hard problems can be used to achieve solutions in polynomial time. The problem considered in this thesis is not so much regarding the reconfiguration of objects, but involves the movement of objects from their current location to a destination, although there is the similarity of moving objects in a constrained space. In moving objects from initial location to final location, the free space in the warehouse need not be constrained, and empty locations may exist in different parts of the warehouse.

Sarrafzadeh and Maddila [13] introduce a discrete warehouse which is a collection of two dimensional unit square objects (robots and obstacles), which are allowed to move only horizontally and vertically along grid lines. The paper considers motion planning problems with movable obstacles. Remote mechanism and contact-based movements are studied and algorithms are proposed to provide solutions for both cases. Similar to the Rush Hour™ problem, the discrete warehouse problem aims to clear a path for the movement of robots in the warehouse as against the work in this thesis, which aims to make a step-by-step movement towards a destination, thus requiring lesser free space in the warehouse.

This thesis seeks to propose an algorithm for the movement of multiple packages in a PDC. The problem has some similarities to sliding block-puzzles, the Rush Hour™ problem and the Warehouseman problem in terms of movement of unit square objects in a confined space, but this research seeks to introduce a priority-based control system for the movement of the
packages. This approach simplifies the control logic and makes for a robust low-level and scalable albeit, non-optimal, control mechanism.
CHAPTER 3: Control Concept

3.1 Introduction

Motion planning is the term used to describe the planning and design of a sequence of discrete steps to achieve movement of objects/robots from one point to another, avoiding obstacles, and negotiating a path to destination. Significant research has been made in the field of motion planning for robots, and algorithms have been proposed to optimize the motion of the robots through a given terrain or layout. The objective of this thesis is to propose a new low-level and robust algorithm for motion planning of loads over automated material handling devices in a public distribution center (PDC). The proposed algorithm uses unique priority values to govern the movement of loads.

3.2 The Layout

For the purpose of this thesis, the layout being considered for the proposed motion planning algorithm is an 8 x 8 square matrix where each of the cells represents a module. The layout can be extended to any number of blocks and the proposed algorithm holds true for any size of the layout. The layout is a planar array of modules over which the loads are transported. The modules are either empty or loaded with packages. The number of empty locations in the layout depends on the occupancy percentage of the warehouse. The layout has loading/unloading bays called docks and these are located on any corner or at any point on
the perimeter of the layout. The proposed layout helps achieve a higher utilization of the available space such that warehouse occupancy can be as high as 80–90% of the total space without adding significant costs. Modules in the layout hold information associated with the load that is placed on them and information that they get from interacting with the modules around them. This layout is reminiscent of a sliding block puzzle, where blocks are placed in some random configuration in a box and the objective is to slide the blocks in order to achieve a final configuration. In the proposed layout the loads are akin to the blocks that are moved around in the warehouse in a series of discrete steps to achieve a final configuration. Figure 1 illustrates the layout of the distribution center.

Figure 1: Representation of DC layout
3.3 Requirements and Assumptions

For the proposed algorithm some requirements have been established and some assumptions are made with regards to the modules, loads and also for motion planning:

1. Modules interact with each other using some technology. The loads are scanned at input and the locations are known at all times.

2. The modules do not have an overall view of the entire system and can interact with only the connected neighboring modules. Modules react to changes in the layout automatically during run time.

3. Module movement is orthogonal in nature. Hence, the loads can only travel in directions perpendicular to the sides.

4. Loads used are unit sized and hence occupy only one module (8.5 x 8.5 in.).

5. All loads in the layout have a unique numerical value for the priority assigned to them. Priority values are maintained by the packages throughout the execution of the algorithm. Choosing the right priority value for each package is beyond the scope of this research.

3.4 The Control Concept

Control of the proposed system is decentralized in nature in that each load/module has autonomy to decide on the actions that it performs within the boundaries of the requirements and assumptions of the system. The characteristics of a module can define the characteristics of the entire system. The control algorithm utilizes object-oriented concepts in that each
module is an object that has attributes and properties, and each of these objects interact with each other to make the system work.

The proposed algorithm is in a sense a greedy algorithm as it is short sighted in its approach, i.e., the loads only look ahead to the next step they wish to take and no future path planning is involved. Any load that wishes to move to a destination will be referred to as an order load hereon. The order load decides on only one-step at a time, and once the movement to the next location is made, the next search is initiated. One important feature of the system is that it is based on a unique numerical value of priority of every load in the PDC, and the algorithm governs the movement of the loads based on these priority levels. The higher priority loads push lower priority loads out of the way to create a path towards their destination. In doing so all the lower priority modules tagged by the module with the order load inherit the priority of the order load and use it to decide the next location for movement.

A module is considered to be in a passive state if it is not loaded, and it is these passive modules which are the “empty spaces” in the warehouse, to which loads are pushed in order to create an “empty space” in the direction of the destination for the selected load. The proposed algorithm is strongly based on object-oriented programming (OOP) concepts wherein each of the modules in the system is an object that holds information and has its own attributes. These attributes are shared in the information sharing process between other modules and data is passed to the control algorithm. The information is used by the control algorithm to enable the order loads to move towards their destinations.
Modules interact with each other and share data in the form of their own attributes as well as that of the loads that the modules carry. Loads are scanned at input and it is assumed for the purpose of this research that a unique priority is set for each of the loads. The priority is a numerical value that forms the basis of movement as order loads with higher priority are given preference for movement to their destination as compared to lower-priority order loads.

A load also has an inherited priority as one of its attributes. The inherited priority of a load is the priority of the order load that is governing motion for other loads. In addition to the information from the loads, modules also have their own attributes like location ID and state.

Table 1 provides a description of the attributes of the modules. The load-module pairs change when movement occurs and every time a new load arrives on a module, the modules need to update their information.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location ID</td>
<td>This is the physical address of the module in the layout</td>
</tr>
<tr>
<td>State</td>
<td>Active/Passive based on whether the module is loaded.</td>
</tr>
<tr>
<td>Priority</td>
<td>Numerical priority of the load on the module, else “0” if not loaded</td>
</tr>
<tr>
<td>Inherited Priority</td>
<td>Numerical value of priority inherited from higher priority module if tagged by module governing motion. If current module is governing motion then Inherited Priority equals Priority.</td>
</tr>
<tr>
<td>Selection Variable</td>
<td>Selection Variable is a numerical value assigned to a load based on its participation in the movement.</td>
</tr>
<tr>
<td>Destination</td>
<td>Location ID of the destination for the selected load</td>
</tr>
<tr>
<td>Next Location</td>
<td>Location ID of the next location.</td>
</tr>
<tr>
<td>Neighbor Location IDs</td>
<td>Location IDs of neighboring modules</td>
</tr>
<tr>
<td>Order Conflict Variable</td>
<td>The order conflict variable is set to true if there is blockage or conflict. Default value is false.</td>
</tr>
</tbody>
</table>
All modules in the system are equipped with the same control logic and each module functions independently of the other modules in the system. The de-centralized control system enables the modules to function like autonomous objects in the constrained space of the layout. The control logic follows a four-step process to execute the algorithm and move to its destination. Modules that carry loads which intend to move to a destination will be called “order loads” from here on. The order loads initiate movement of other loads so as to push them out of the way to vacate the next module, thus enabling the order load to move one step closer to the destination. The control logic is a four-step process that includes gathering information from the loads, sharing that information with modules, selecting the next locations for motion and executing motion. The order loads follow all the four steps of the control logic whereas the other loads use only a part of the control logic. All the modules in the layout have the capability of governing motion and communicating with modules around them. Figure 2 shows the control logic as a series of steps that are followed by the modules in that order.

![Diagram of the control logic]

**Figure 2 : Overview of the control logic**
The first step of the algorithm is the identification process, whereby the module checks if it is loaded i.e., in an active state. If loaded, then the module retrieves the priority value of the load placed on it. Some of the other load variables that the module retrieves are the *Inherited Priority*, *Selection Variable*, and the *Destination* of the load it handles. These data are required by the module in the next steps of the control algorithm. Each of the modules functions as an object in an Object-Oriented environment and holds the collected information as its own attributes. The *Current Location* is unique for each module and is stored in the module. The *Selection Variable* for loads is set in the next steps when modules are tagged for next location. Figure 3 shows the identification process for modules.

![Identification process for modules](image)

**Figure 3 : Identification process for modules**

After identification, if the module is loaded, then it interacts with the neighboring modules and retrieves information pertaining to their load variables. The information sharing between modules is always in real time and needs to be updated in each time step. In the retrieval
stage, the module also determines its neighbors and communicates with all the modules that are its immediate neighbors. The information from neighboring modules consists of Location ID, State, Priority, Inherited Priority, and Selection Variable. Following the first two steps, which are primarily for data collection, the third step is the decision on the next location. Only the order loads get the choice of next locations and hence this step is used only for the order loads. For the non-order loads, their next location is decided by any order load that governs its motion. Once the module gets information from the load it carries, as well as of the neighboring modules, the control algorithm uses comparison of properties as a way of narrowing down the list of feasible options between the neighbors. The neighbors are shortlisted based on the distance from destination, the modules with minimum distance to destination are chosen. As a rule, this algorithm does not allow any lower-priority module to tag or select a higher-priority neighbor. The neighbor locations with higher priority than the current location are rejected. Out of the feasible options, the neighbor with the lowest priority is chosen as the next location. When no neighbor location is feasible, the load is blocked for the current time step. This procedure is called the Next Location Search. Figure 4 (A) illustrates the Next Location Search. Figure 4 (B) shows the naming convention used for modules throughout this thesis.
Once a next location is decided, the algorithm checks if the module at the next location is in active or passive state. If the next location is empty, the load at the current location only has to move to the next location. The movement process is explained in the subsequent pages. In the alternative case of the next location being occupied by a load, the algorithm needs to move the package at the next location out of the current module. The module carrying the order load initiates an iterative process of tagging lower priority modules amongst its neighbors until an empty location is found. This is a heuristic approach designed to use the core principle of this algorithm. The higher-priority loads push the lower-priority loads out of their way in a bid to get to the destination faster. The tagging process ensures that the order load passes on its priority to the tagged modules, indicating its hold over the resources until the movement sequence is complete. The tagged modules would ultimately work as a series
of orthogonal translations (movements to vacate the module at the next location for the order load to move. Figure 5 shows the iterative process of tagging modules.

![Flowchart for Tagging Modules](chart.png)

**Figure 5**: Tagging modules for movement

The process of tagging modules also involves a search procedure, which is similar to the *Next Location Search*. However, the search procedure for tagging the modules does not depend on the distance to destination and thus provides more neighbors to choose from as the potential next location to be tagged. The algorithm uses comparison of properties as a way of narrowing down the list of feasible options to choose between the neighbors. The neighbor locations with lower priority than the inherited priority of the current location are selected as potential next locations. The algorithm also checks if the neighbor has already been tagged by some other module, in which case the *Selection Variable* of the neighboring module (load) is “1” and such modules are not considered as potential next locations. The neighbor
locations satisfying both of the mentioned conditions are the feasible next locations. Out of the feasible options, the neighbor with the lowest priority is chosen. The failure to find a next location results in the load being blocked and all the tagged loads to be reset. The load remains blocked until such time that the search procedures do not find an empty location in subsequent time steps. In cases where any of the neighbors shortlisted as feasible next locations are in a passive state (empty), such neighbors are preferred as the next location. Once a module is tagged as the next location, then it in turn searches for the next location and this iterative search procedure stops only when some module encounters an empty module as its next location. The common property between all the tagged modules is the inherited priority; all the tagged modules inherit the priority of the order load that initiates this procedure. Figure 6 details the Tagging Modules for movement process. The values at the bottom right corner indicate the location IDs; the values in the small squares at the top center indicate the inherited priorities and the numerical values at the center show the priority values of each module.

Figure 6 : Illustration of the Tagging Modules for Movement process
Module 2 with a priority level of 4 and an inherited priority level of 8, searches for a potential next location. Module 3 (Case I) is tagged as next location as the priority of module 3 is lesser than the priority of module 1. Since another load occupies Module 3 having priority 1, hence Module 3 will start the search for the next location to be tagged and the only option it has is module 6. The inherited priority of module 3 is “8” which is lesser than the priority of the module 6 which is “9”. In such a case the module tagging process is interrupted. Since no empty location has been found, the algorithm then backtracks and at Module 2, chooses the only other available module for motion (Case II). Now module 1 is chosen as the next location since it satisfies all criteria in the selection process. For Case II, It can be seen that if module 1 is picked, then a sequence of modules to be tagged can be generated to find an empty location at the bottom right corner. All modules tagged for case II satisfy the criteria that the priority of the potential next location need be lesser than the inherited priority of the module searching for next location.

The last step in the control logic is moving the loads over the modules. The Movement process begins when the modules tagging process finds an empty location. As described in Section 3.3, the movement is orthogonal, and the algorithm supports block movement of loads, i.e., moving the loads on these modules takes more than one time step, as the change of orientation in movement would result in some loads having to wait and suffer a delay of one time step before they can move. Figure 7 shows that load at module 4 needs to wait for one time step, whereas loads at module 1, 2, and 3 can move together in one time step. There is a time delay of one time step before the next movement. This delay caused due to the
movement process leads to different order loads having to wait for a varying amount of time before moving to their next location. This delay depends only on the number of modules tagged for movement and may change in case of a conflict. Conflicts and blockages are discussed in the next section.

![Diagram showing movement into time steps](image)

**Figure 7: Movement of loads**

The same procedure is used in the example in Figure 8 where for moving the load at module 5 to module 2, a series of orthogonal translations over 4 time steps is required. All loads travelling in the same direction move together in one time step and the remaining modules in the tagged sequence will move in the future time steps depending on the orientation of movement. Table 2 shows breaking of a movement sequence over four time steps.

<table>
<thead>
<tr>
<th>Time Step</th>
<th>Tagged Modules</th>
<th>Modules participating in movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>t = 1</td>
<td>5, 2, 1, 4, 7, 8, 9</td>
<td>7, 8, 9</td>
</tr>
<tr>
<td>t = 2</td>
<td>5, 2, 1, 4, 7</td>
<td>1, 4, 7</td>
</tr>
<tr>
<td>t = 3</td>
<td>5, 2, 1</td>
<td>1, 2</td>
</tr>
<tr>
<td>t = 4</td>
<td>5, 2</td>
<td>5, 2</td>
</tr>
</tbody>
</table>
When a load moves from one module to another the properties of the modules change and the modules constantly need to update the information they carry about the loads and the state they are in. The real time updating of information is crucial to the process of movement and also to the search of next locations. The proposed algorithm uses the steps described in this section to enable movement of multiple loads in a warehouse simultaneously. The movement of multiple loads makes the system more complex as conflicts arise for accessing the same

Figure 8: Illustration of load movement
resource (module) and these conflicts need to be resolved during run time. The following section explains the process of moving multiple loads simultaneously to their destinations.

3.5 Simultaneous Movement of Multiple Packages: Conflicts & Blockages

The proposed algorithm can be used for simultaneous movement of multiple loads to multiple destinations using a list of the order loads called the Order List. The order list is a list of all Selected Loads in decreasing order of priority. The order list executes the control algorithm for each load in the list. In each time step all the loads in the order list undergo some processing as per the control logic. The key advantage that this system provides is the sequential processing of all loads wanting to move to some destination at every time step. In addition, at every time step, the load with the highest priority gets to choose modules for movement to its next location and Order Loads are processed in decreasing order of priority. The difference between moving a single package and multiple packages is the possibility of conflicts in the load movement. The proposed algorithm factors in some conflict resolution that is not too complex in implementation and is in tune with the low level nature of the algorithm. In order to explain the conflicts and blockages and the built-in resolution methods, we will assume a layout similar to the one discussed in Section 3.2.

Case I: Two Order Loads trying to access the same module for movement

As shown in Figure 9, a conflict arises when more than one order load tries to access the same resource for movement. This is solved by using the Order List. The order list provides
the first access of modules to the higher priority load and once a particular load has chosen a set of locations, they become unavailable for other order loads until such time their motion is complete.

![Figure 9: Multiple loads accessing the same resource](image)

As is shown by Figure 9, once a module is chosen by an order load as its next location, the module is blocked for all other loads trying to access it for movement. The next location module is then available for motion only when movement of load with priority 9 is complete.

**Case II: Higher priority order load tags a lower priority order load**

The second case for conflicts arises when a higher priority order load tags a lower priority order load for movement. This creates a problem as both order loads have their own destinations and trying to move to destination in the fewest possible steps. In a situation where a higher priority order load say X tags a lower priority order load, say Y, the lower priority order load Y then behaves like any other normal package and is moved according to the movement sequence generated by X. It is only after the load X releases its hold on Y, that the load Y can re-start its movement to its destination. Figure 10 shows the delay in the
processing of a lower priority order load when a higher priority order load tags it for movement. As shown, in the time step 1, the order load 6 does not hold its properties of an order load and is not processed for its movement closer to its destination. Rather, it behaves like a normal package to allow the higher priority order load 7 to execute its movement. It is only in the time step 2 that the Order Load 10 (earlier 6) is processed.

**Figure 10**: Higher priority order load tagging a lower priority order load

In such kinds of conflicts, the *Next Location* for the lower priority load gets reset and the modules tagged by the lower priority load also get reset. In this case the next location for
Order Load 2 is reset when it is tagged for movement by Order Load 1. Once Order Load 2 has executed its movement for Order Load 1, the hold is released and Order Load 2 will have to reinitiate the search for its next location and subsequently tag other modules for movement.

Case III: Lower priority Order Load blocked by higher priority load at destination

This conflict occurs specifically in situations when all the destinations are grouped together and a higher priority order load, say X, gets to destination faster than a lower priority order load, say Y. The lower priority load gets blocked while trying to get to its destination.

![Figure 11: Lower priority order load blocked while getting to destination](image)

Figure 11 shows the blockage caused due to the Order Load 1 being blocked for movement to destination 3 because a higher-priority load at location 2 is at destination. The load at 1 is unable to select module 5 as next location since it is not the module with the minimum distance to destination. To solve this blockage, the algorithm will assign a temporary destination to the blocked module. The temporary destination is assigned in such a way that it
is closer to the original destination of the blocked module but allows the current blockage to be released. The assignment of the temporary destination is done automatically after the module checks two conditions and if the two conditions are met then a temporary destination is assigned to the blocked load.

The following two conditions are checked:

1. All loads higher than the blocked load need to be at their destinations.
2. The temporary destination is not the destination for any other order load.

![Figure 12: Assignment of a temporary destination to blocked order load](image)

Figure 12 shows the assignment of a temporary destination to the blocked order load. In the figure, location 7 is assigned as the temporary destination and hence the blocked load at 1 can move towards the temporary destination 7 by choosing location 5 as it next location. Once Order Load 2 reaches its temporary destination, the algorithm re-sets the original destination from memory and the Order Load can move towards its original destination without being blocked. Using the control concept explained in this chapter, the algorithm can be used to successfully negotiate the movement of any number of order loads to their
destinations. With the current design of the control system, the possibility of a complete “lock up” of packages is eliminated. The highest priority package always manages to get its way to its destination provided there is at least one empty space for facilitating movement.

The following chapter details the control algorithm and presents some of the results of the algorithm.
CHAPTER 4: Priority-based Control Algorithm

4.1 Introduction

The complete priority-based control algorithm is presented in this chapter, along with an example of the package movement scenario and a summary of results. As discussed in the last chapter, the control algorithm can be divided into three main procedures: the main section that controls the motion of all the order loads, the subsection which helps the order loads decide on the next location for movement, and the subsection that tags modules involved in movement of the order load from the current location to the next location.

4.2 The Control Algorithm

The main body of the algorithm is the implementation of the first, second and fourth steps of the control logic explained in Section 3.4. The steps of identification, communication, information retrieval and movement all form a part of the main body. The third step of searching for a next location and subsequently tagging modules for moving the Order Load at current location to the next location is shown as sub process 1 and sub process 2. Sub process 1 is the procedure for searching for Next Location and sub process 2 details the module tagging procedure. On the following pages, a step-by-step procedure is explained for the main body of the algorithm followed by Figure 15 and Figure 16. These are the flow charts of the two sub processes, Next Location Search and Tagging modules for movement. The main body of the algorithm is explained in the form of a pseudo code in Figure 13.
Procedure
Get Order List
While All Order Loads not at destinations
  For each Order Load
    If Order Load at destination then
      Skip Order Load for processing.
    End If
    If Order Load conflict then
      Skip Order Load for processing (Order Load blocked) and Reset tagged loads
    End If
    If Order Load blocked by higher priority Order Load at destination then
      If All Order Loads with higher priority at destinations then
        Assign a temporary destination to current Order Load
      Else
        Skip Order Load for processing (Order Load blocked) and Reset tagged loads
      End If
    End If
    If Order Load at temporary destination then
      Reset destination as original destination
    End If
    If Movement sequence from last time step complete then
      Next Location Search, Tag Modules for Movement
      Execute Movement as per new sequence.
    Else
      Execute Movement sequence pending from last time step
    End If
  End For
  Update $t = t + 1$
End While. Report Retrieval Time $t$.
End Procedure

Figure 13 : Pseudo code for algorithm
The first step is the information gathering process, where the control algorithm retrieves the location ids of all order loads, and also the priorities of packages in the layout. Order List processing starts at time $t = 0$. The algorithm checks if all the Order Loads are at their destinations. If the check condition returns true, then no further processing is required and the algorithm returns the retrieval time $t$. If the check condition returns false, the algorithm continues with processing the Order List. Only the Order Loads that are not at destinations will be processed and the processing is a recursive process until such time all Order Loads are at destinations. For each of the Order Loads, the following check points are put in place to determine the participation of the Order Load in the current time step. Next step is checking if there has been an Order Load conflict for the current Order Load. An Order Load conflict occurs when a lower priority Order Load is tagged by a higher priority Order Load for movement. In such cases, the lower priority Order Load ceases to behave as an Order Load and waits till it has moved as per the sequence of the higher priority Order Load, and then resumes its motion towards its destination. In case of an Order Load conflict, the lower priority order load is skipped for processing in the Order List and any modules tagged by the lower priority Order Load in the past time steps are reset. The next check point in the sequence is to check if any of the lower priority Order Loads is blocked due to a higher priority Order Load at destination. This kind of a conflict is explained in detail in Section 3.5 as Figure 11. In such cases a temporary destination needs to be assigned to the lower priority load so that a new route can be negotiated since the lower priority Order Load cannot push a higher priority Order Load. The algorithm will report a blockage, reset any tagged modules from the previous time step and then assign a temporary destination to the Order Load. A
temporary destination can only be assigned if all the Order Loads having a higher priority as compared to the blocked load are at their destinations. The temporary destination is set in a way where it takes the lower priority Order Load around the higher priority Order Load which is at destination. The next processing step is to check for the Order Load to be present at its temporary destination in the current time step. If yes, then the original destination for the Order Load is restored enabling the Order Load to move to its original destination from the next time step. Once the initial checks are complete, the algorithm needs to check if the current Order Load has pending movement sequence from the last time step; in such a case the Order Load has a next location already decided and it will not be considered for the search process explained until the movement sequence has been executed successfully. Only when the movement sequence from a past time step is complete or in case of conflicts and blockages leading to resetting of tagged modules, can the Order Load be considered for the Next Location Search and Tagging Modules for Movement process.

If the Order Load has completed its movement sequence from the past time steps and is not blocked then it is allowed to initiate search for the next location and tag modules for movement. The Order Loads that have been assigned temporary locations and those at temporary destinations are also considered for this step. The two processes of Next Location Search and Tagging Modules for Movement are explained in detail in Figure 15 and Figure 16 on the following pages. If any of the two sub processes are unable to find a next location and report a blockage, then the processing for the current Order Load is skipped for this time step and all the tagged modules are reset. This step is the last step in the decision making
process and now movement is executed as per the procedure explained in the last chapter, i.e., depending on the movement orientation; the sequence is executed over multiple time steps. In the movement step, movement sequence pending from the last time step is used or if a new search was executed successfully in the current time step, then the new movement sequence is used. Only movement for the current time step is carried out in this step, the other tagged modules retain their inherited priorities and will move in the future time steps. The procedure explained above is repeated for all Order Loads eligible for processing within the Order List. Processing for one time step $t$ is complete. Movement occurs for all Order Loads simultaneously in each time step. The procedure is repeated until all Order Loads are at destinations and retrieval time $t$ is reported.

On the following pages the sub processes of Next Location Search and Tagging Modules for Movement are explained in Figure 15 and Figure 16. Figure 14 shows the Key to the charts.

![Key to the charts](image)

**Figure 14 : Legend for the flowcharts**
Figure 15: Searching for Next Location - Sub process 1
TAGGING MODULES FOR MOVEMENT

From Current Load, Next Location, Destination, Priority

Retrieve Information

SET Current Location

Generate List of Neighbors

Iterate Over List of Neighbors

Remove Neighbor from list of feasible neighbors

Current Location Inherited Priority > Next Location Priority

YES

Neighbor already tagged by higher priority module

YES

Add to list of Possible Next Locations

Set Current Location Priority as Next Location Inherited Priority

YES

SET Selection Variable = 1

Pick the location with lowest priority from the list of possible next locations

NO

Module Movement Blocked. No feasible next location

NO

Set Order Conflict Variable of Next Location as True

NO

Next Location contains an order load

YES

Next Location = Empty

YES

Return To Main Algorithm

SET Next Location as Current Location and Continue Search

Figure 16: Tagging modules for movement - Sub Process II
4.3 Scenario I

To illustrate the working of the algorithm, two scenarios will be considered. The first scenario is a 6 x 6 configuration with four randomly placed order loads in the layout. The destinations/ docks for these loads are placed at 4 corners. The chances of a lower priority load being pushed away from its destination are very high in this kind of a layout, as all the four order loads move in different directions. Figure 17 illustrates Scenario I.

![Figure 17: Scenario I for priority based control algorithm](image)

For the example in Figure 18, a 6 x 6 layout is considered for easier explanation. At t = 0, the order placement is shown along with the destinations for each of the order loads. There are 4 order loads with priorities 33, 32, 31, 30 placed at locations 24, 15, 33, 12. It is seen that at time t = 1, the Order Load 4 is blocked for movement. The priority for this load is 30, which is the lowest priority order load. In order to get to its destination (Location 36), the only
feasible location it can move to, is location 7 which is being used by Order Load 1 with a higher priority. Hence Order Load 4 remains blocked for the first time step. The Order Load 3 makes a move towards its destination whereas the other two Order Loads initiate movement of some loads in the time step $t = 1$. Order List is maintained by Location IDs.

**Figure 18 : Scenario I - time step $t = 0$ and time step $t = 1$**
Figure 19 shows the time step $t = 2$ and time step $t = 3$. Order Load 1 moves one step closer to its destination in time step $t = 3$ whereas Order Load 2 still has other loads moving out of their current module to clear the chosen next location. Order Load 3 reaches its destination at time $t = 2$ and it is observed that the Order Load 4 is blocked at the end of time steps 2 and 3.

**Figure 19 : Scenario I - time step $t = 2$ and time step $t = 3$**
Figure 20 shows the time step 4 and 5. Order Load 1 tags new loads for movement and some loads execute movement as per sequence. Order Load 2 moves one step closer to destination from location 9 to 10. Order Load 3 remains at destination and will do so till it is pushed out by a higher priority load. Order Load 4 continues to be blocked because of Order Load 1.

**Figure 20 : Scenario I - time step t = 4 and time step t = 5**
Figure 21 shows the time step 6 and 7. At time $t = 6$, Order Load 1 moves a step closer to its destination by moving from location 18 to 17 also at $t = 7$, new loads are tagged and block movement is observed. Order Load 2 has loads moving for it at $t = 6$ and gets a step closer to destination at time $t = 7$. Order Load 4 is chosen by Order Load 1 at $t = 7$ and hence there is an order load conflict.

**Figure 21 : Scenario I - time step $t = 6$ and time step $t = 7$**
Figure 22 shows the time step 8 and 9. For Order Load 1 tagged modules execute movement as per sequence, block movement is observed at \( t = 9 \). Order Load 2 is blocked for movement at \( t = 8 \) and \( t = 9 \) due to Order Load 1 having a hold on its potential next location. Order Load 4 is now relieved of the hold and hence tags modules for movement, owing to block movement, it moves a step closer to its destination at \( t = 8 \) and tags new modules at \( t = 9 \).

---

**Figure 22 : Scenario I - time step \( t = 8 \) and time step \( t = 9 \)**
Figure 23 shows the time step 10 and 20. For the example provided here there are no more conflicts after time step 10 and the all the 3 remaining order loads make their way towards their destination by using the algorithm. At \( t = 20 \) all the order loads reach their intended destinations and all loads in the layout become free of any inheritance from the order loads.

Following the same step wise procedure described in the algorithm and in this example, At \( T = 20 \), the final desired configuration of packages is achieved. On the left is the final configuration of packages
4.4 Scenario I - Testing

The proposed algorithm was executed for 1200 iterations for scenario I on an 8 x 8 configuration at varying levels of warehouse densities to calculate the average number of time steps required for moving all loads to their destinations. The data collected from 300 observations is included in the Appendix. The performance is measured in terms of retrieval time steps. The retrieval times are the total number of time steps required to move all the Order Loads to their destinations. Figure 24 shows the retrieval time (steps) against the density of the warehouse. Based on the data collected, the average retrieval time was calculated as the average of all retrieval times for a particular warehouse density. It is seen that there is big increase in the average retrieval time when the density increases from 90% to almost 100%. For lower utilization density, the increment in time steps is almost linear.

![Retrieval Time vs Density](image)

**Figure 24 : Performance of algorithm for scenario I**
4.5 Scenario II

The second scenario to be considered is also a 6 x 6 configuration with four randomly placed order loads in the layout. The destinations/docks for these loads are together at the center of the top two rows. The chances of a conflicts increase in this layout as the loads move in the same direction, towards the destinations. Figure 25 illustrates the second scenario.

![Figure 25: Scenario II for priority based control algorithm](image)

For the example in Figure 26, a 6 x 6 layout is considered for easier explanation. At $t = 0$, the order placement is shown along with the destinations for each of the order loads. There are 4 order loads with priorities 36, 35, 32, 31 placed at locations 16, 14, 17, 25 respectively. The destinations for these loads in their order are 3, 4, 9, and 10. Figure 26 through to Figure 33
explain the 14 time steps that lead from the initial configuration to the final desired configuration with all order loads at their destination. The example also provides an insight into the conflict resolution procedure of assigning a temporary destination to a blocked order load. At time $t = 1$, Order Load 1 moves one step closer to destination. Order Load 2 has a block movement spanning two modules i.e. Loads 14 and 8, move simultaneously in the same time step to locations 8 and 2. Order Loads 3 and 4; also closer to their destinations.

Figure 26 : Scenario II - time step $t = 0$ and time step $t = 1$
At time $t = 2$, all the order loads again move a step closer to destination as shown in Figure 27. At $t = 3$, Order Load 3 is blocked due to Order Load 1 moving into the potential next location for Order Load 3. Order Load 2 initiates movement of tagged loads whereas Order Load 4 makes a movement towards its destination.

**Figure 27: Scenario II - time step $t = 2$ and time step $t = 3$**
Figure 28, at time $t = 4$, Order Load 1 reaches its destination and blocks Order Load 2 and 3 for motion. Order Load 4 moved to location 15 and is a step closer to its destination. At time $t = 5$, Order Load Conflict resolution process assigns a temporary destination to Order Load 2 so as to clear the block. Meanwhile Order Load 3 reaches its destination.

Figure 28: Scenario II - time step $t = 4$ and time step $t = 5$
Figure 29 shows the time steps 6 and 7. At $t = 6$, Order Load 3 still remains at destination. Order Load 2 starts moving towards its new temporary destination. Order Load 3 and 4 reach their destination. At time $t = 7$, The Order Load 2 moves another step towards its temporary destination.

![Diagram of Scenario II at time steps 6 and 7]

**Figure 29**: Scenario II - time step $t = 6$ and time step $t = 7
Figure 30 shows the time steps 8 and 9. At $t = 8$, Order Load 2 moves yet another step towards its destination while all other Order Loads remain at their destination. At time $t = 9$, Order Load 2 reaches the temporary destination set at $t = 5$ and hence the algorithm resets the original destination for Order Load 2 and thus in the subsequent time steps Order Load 2 will move towards location 4.

**Figure 30 : Scenario II - time step $t = 8$ and time step $t = 9$**
In Figure 31 at $t = 10$, Order Load 2 pushes Order Load 4 out of its destination to make way for itself to reach its original destination. At $t = 11$, The Order Load 2 moves to location 10 and Order Load 4 still remains blocked.

Figure 31: Scenario II - time step $t = 10$ and time step $t = 11
Figure 32, at $t = 9$, The Order Load 2 initiates block movement for loads at location 4 and 5 to locations 5 and 6 respectively. This movement clears the destination for Order Load 2. Order Load 4 however still remains blocked. At $t = 13$, Order Load 2 moves to its destination.

<table>
<thead>
<tr>
<th>Time T=12</th>
<th>Time T=13</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Figure 32: Scenario II - time step t = 11 and time step t = 12" /></td>
<td></td>
</tr>
</tbody>
</table>

55
Figure 33 shows the final configuration at $t = 14$. Order Load 4 moves to its destination at $t=14$. Order Load 2 moved to destination at the end of time step $t = 13$ and Order Load 1 and 2 were already at destination. The load movement is complete with all the loads reaching their destinations and no blockages.

<table>
<thead>
<tr>
<th>TIME T=14</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 1 2 36 35 18 5 28</td>
</tr>
<tr>
<td>30 7 8 32 31 10 11 12</td>
</tr>
<tr>
<td>0 13 14 15 16 17 18</td>
</tr>
<tr>
<td>0 19 20 21 22 23 24</td>
</tr>
<tr>
<td>0 25 26 27 28 29 30</td>
</tr>
<tr>
<td>0 31 32 33 34 35 36</td>
</tr>
</tbody>
</table>

Order List : [3, 4, 9, 11]

Load 3 is at Destination
Load 4 is at Destination
Load 9 is at Destination

Next Location for Load : 11 is : 10
Tagged Modules for Load : 11 is : [11, 10]
Movement in Current Iteration: [11, 10]

Figure 33 : Scenario II - final configuration
4.6 Scenario II - Testing

The proposed algorithm was run for 1200 iterations for scenario II as well, on an 8 x 8 configuration at varying level of warehouse densities to calculate the average number of time steps required for moving all loads to their destinations. The data collected from 300 observations is included in the Appendix. Figure 34 shows the retrieval time (steps) against the density of the warehouse. The retrieval times are the total number of time steps required to move all the Order Loads to their destinations. Based on the data collected, the average retrieval time was calculated as the average of all retrieval times for a particular warehouse density. It is seen that there is big increase in the average retrieval time when the density increases from 90% to almost 100%. For lower utilization density the increment in time steps is fairly linear. The algorithm presented in this section works for any configuration and number of packages moving to destinations.

![Retrieval Time vs Density](image)

Figure 34 : Performance of Algorithm for Scenario II
CHAPTER 5: Conclusions and Future Work

This report presents a new priority-based control algorithm to enable movement of packages in the public distribution center. The algorithm along with its implementation in Java seeks to use an object-oriented methodology and apply it to the material handling system in a public distribution center. Each of the small handling devices functions as objects and the algorithm works by using the information stored and shared by these objects.

The approach to the package movement problem proposed in this thesis can be defined as a decentralized, greedy approach that focuses on getting packages one step closer to their final destination in each time step. The algorithm also uses a brute force approach wherein the movement sequences are generated using a simple logic requiring high number of iterations. The algorithm only looks at the immediate next step without any planning over the future time steps. Conflicts and blockages are resolved by using the unique priority values of the packages, and the algorithm resolves conflicts with relative ease.

The proposed algorithm meets its objective of being a simple, low level yet robust control mechanism that has the flexibility to be adapted to different configurations. The work presented in this report is expected to be preliminary work for building an optimal and a more sophisticated control system in the future.
The modular implementation allows for future improvements in areas such as *Next Location Search* or *Tagging Locations for Movement*. These processes can be improved individually without impacting the algorithm as a whole. Future work may be aimed at optimizing some of the search procedures used in the algorithm. Following are some topics for future research.

**Next Location Search**

In the algorithm presented in this report, a next location is chosen based on distance to destination of the potential next locations and then the lower priority location is picked as the next location. This is one area where future research may focus on choosing the next location by looking into the future time steps and choosing the location that may enable movement with the lesser chances of blockage. This can be achieved by scanning more than one neighboring location in each direction and identifying the direction providing the least resistance for future movement.

**Tagging Modules for Movement**

The tagging process is another area of future research. The process is essentially used for finding an empty location in the layout and then tagging modules in an order such that loads can be pushed into the empty location and an empty location can thus be created in the direction of movement, next to the current location. The proposed algorithm provides a non-optimal method of tagging modules and future research may focus on introducing algorithms like the A* or Dijkstra's algorithm for finding empty locations within the layout and then
tagging modules to start movement towards it. A Wave search algorithm can be used to look for empty locations in place of the simple heuristic approach that is being used in this report.

**Higher Level of Control for Movement**

The proposed control system is a low-level method of control where no future planning is used. The conflict resolution is also at low-level. Introducing a higher, more sophisticated level of control for movement and resolution of conflicts is another potential area of research. The higher level of control may use priorities of packages as a way of better controlling the movement process. Priorities may be dynamically reassigned to solve conflicts faster and in a more efficient way to reduce the retrieval times.
REFERENCES


[10] Hearn, Robert A. and Demaine, Eric D., PSPACE-completeness of sliding-block puzzles
and other problems through the nondeterministic constraint logic model of computation, 2005, Theoretical Computer Science, 343, 72–96


APPENDICES
## Appendix A: Scenario I – Algorithm Performance

<table>
<thead>
<tr>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>117</td>
<td>69</td>
<td>19</td>
<td>116</td>
<td>67</td>
<td>31</td>
<td>100</td>
<td>95</td>
<td>54</td>
<td>94</td>
<td>79</td>
<td>36</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>72</td>
<td>24</td>
<td>31</td>
<td>25</td>
<td>71</td>
<td>32</td>
<td>100</td>
<td>73</td>
<td>46</td>
<td>72</td>
<td>23</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>77</td>
<td>1</td>
<td>33</td>
<td>35</td>
<td>76</td>
<td>33</td>
<td>100</td>
<td>98</td>
<td>9</td>
<td>75</td>
<td>44</td>
<td>97</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>58</td>
<td>57</td>
<td>37</td>
<td>19</td>
<td>32</td>
<td>34</td>
<td>100</td>
<td>108</td>
<td>46</td>
<td>27</td>
<td>107</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>99</td>
<td>61</td>
<td>38</td>
<td>29</td>
<td>98</td>
<td>35</td>
<td>100</td>
<td>61</td>
<td>52</td>
<td>30</td>
<td>5</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
<td>95</td>
<td>69</td>
<td>94</td>
<td>30</td>
<td>54</td>
<td>36</td>
<td>100</td>
<td>71</td>
<td>1</td>
<td>29</td>
<td>45</td>
<td>70</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>73</td>
<td>16</td>
<td>23</td>
<td>43</td>
<td>72</td>
<td>37</td>
<td>100</td>
<td>62</td>
<td>57</td>
<td>30</td>
<td>61</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>96</td>
<td>47</td>
<td>30</td>
<td>77</td>
<td>95</td>
<td>38</td>
<td>100</td>
<td>66</td>
<td>26</td>
<td>2</td>
<td>65</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
<td>54</td>
<td>42</td>
<td>53</td>
<td>32</td>
<td>26</td>
<td>39</td>
<td>100</td>
<td>87</td>
<td>38</td>
<td>25</td>
<td>86</td>
<td>57</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>62</td>
<td>33</td>
<td>21</td>
<td>26</td>
<td>61</td>
<td>40</td>
<td>100</td>
<td>65</td>
<td>64</td>
<td>61</td>
<td>61</td>
<td>35</td>
</tr>
<tr>
<td>11</td>
<td>100</td>
<td>53</td>
<td>40</td>
<td>52</td>
<td>23</td>
<td>32</td>
<td>41</td>
<td>100</td>
<td>74</td>
<td>66</td>
<td>9</td>
<td>11</td>
<td>73</td>
</tr>
<tr>
<td>12</td>
<td>100</td>
<td>74</td>
<td>21</td>
<td>51</td>
<td>60</td>
<td>73</td>
<td>42</td>
<td>100</td>
<td>62</td>
<td>18</td>
<td>61</td>
<td>4</td>
<td>35</td>
</tr>
<tr>
<td>13</td>
<td>100</td>
<td>113</td>
<td>67</td>
<td>64</td>
<td>112</td>
<td>5</td>
<td>43</td>
<td>100</td>
<td>72</td>
<td>43</td>
<td>26</td>
<td>71</td>
<td>28</td>
</tr>
<tr>
<td>14</td>
<td>100</td>
<td>84</td>
<td>71</td>
<td>36</td>
<td>83</td>
<td>27</td>
<td>44</td>
<td>100</td>
<td>65</td>
<td>64</td>
<td>34</td>
<td>59</td>
<td>20</td>
</tr>
<tr>
<td>15</td>
<td>100</td>
<td>63</td>
<td>36</td>
<td>62</td>
<td>39</td>
<td>52</td>
<td>45</td>
<td>100</td>
<td>60</td>
<td>59</td>
<td>29</td>
<td>35</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>100</td>
<td>69</td>
<td>42</td>
<td>43</td>
<td>55</td>
<td>68</td>
<td>46</td>
<td>100</td>
<td>65</td>
<td>64</td>
<td>1</td>
<td>48</td>
<td>43</td>
</tr>
<tr>
<td>17</td>
<td>100</td>
<td>68</td>
<td>46</td>
<td>67</td>
<td>37</td>
<td>52</td>
<td>47</td>
<td>100</td>
<td>64</td>
<td>63</td>
<td>28</td>
<td>46</td>
<td>22</td>
</tr>
<tr>
<td>18</td>
<td>100</td>
<td>92</td>
<td>61</td>
<td>72</td>
<td>91</td>
<td>58</td>
<td>48</td>
<td>100</td>
<td>67</td>
<td>66</td>
<td>58</td>
<td>13</td>
<td>63</td>
</tr>
<tr>
<td>19</td>
<td>100</td>
<td>70</td>
<td>38</td>
<td>25</td>
<td>69</td>
<td>24</td>
<td>49</td>
<td>100</td>
<td>78</td>
<td>41</td>
<td>62</td>
<td>16</td>
<td>77</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>112</td>
<td>1</td>
<td>111</td>
<td>71</td>
<td>20</td>
<td>50</td>
<td>100</td>
<td>134</td>
<td>42</td>
<td>5</td>
<td>133</td>
<td>117</td>
</tr>
<tr>
<td>21</td>
<td>100</td>
<td>87</td>
<td>37</td>
<td>70</td>
<td>86</td>
<td>35</td>
<td>51</td>
<td>100</td>
<td>104</td>
<td>44</td>
<td>65</td>
<td>64</td>
<td>103</td>
</tr>
<tr>
<td>22</td>
<td>100</td>
<td>91</td>
<td>69</td>
<td>35</td>
<td>78</td>
<td>90</td>
<td>52</td>
<td>100</td>
<td>78</td>
<td>77</td>
<td>2</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>23</td>
<td>100</td>
<td>94</td>
<td>93</td>
<td>26</td>
<td>13</td>
<td>60</td>
<td>53</td>
<td>100</td>
<td>92</td>
<td>75</td>
<td>26</td>
<td>91</td>
<td>42</td>
</tr>
<tr>
<td>24</td>
<td>100</td>
<td>120</td>
<td>24</td>
<td>28</td>
<td>119</td>
<td>65</td>
<td>54</td>
<td>100</td>
<td>85</td>
<td>32</td>
<td>70</td>
<td>84</td>
<td>58</td>
</tr>
<tr>
<td>25</td>
<td>100</td>
<td>171</td>
<td>58</td>
<td>14</td>
<td>73</td>
<td>170</td>
<td>55</td>
<td>100</td>
<td>101</td>
<td>44</td>
<td>19</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>26</td>
<td>100</td>
<td>40</td>
<td>14</td>
<td>38</td>
<td>39</td>
<td>1</td>
<td>56</td>
<td>100</td>
<td>86</td>
<td>85</td>
<td>70</td>
<td>37</td>
<td>44</td>
</tr>
<tr>
<td>27</td>
<td>100</td>
<td>56</td>
<td>33</td>
<td>20</td>
<td>55</td>
<td>53</td>
<td>57</td>
<td>100</td>
<td>79</td>
<td>33</td>
<td>78</td>
<td>56</td>
<td>49</td>
</tr>
<tr>
<td>28</td>
<td>100</td>
<td>77</td>
<td>25</td>
<td>76</td>
<td>33</td>
<td>55</td>
<td>58</td>
<td>100</td>
<td>79</td>
<td>74</td>
<td>42</td>
<td>78</td>
<td>62</td>
</tr>
<tr>
<td>29</td>
<td>100</td>
<td>37</td>
<td>36</td>
<td>22</td>
<td>33</td>
<td>21</td>
<td>59</td>
<td>100</td>
<td>50</td>
<td>25</td>
<td>29</td>
<td>49</td>
<td>22</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>86</td>
<td>59</td>
<td>53</td>
<td>74</td>
<td>85</td>
<td>60</td>
<td>100</td>
<td>91</td>
<td>12</td>
<td>90</td>
<td>50</td>
<td>28</td>
</tr>
</tbody>
</table>

### Table 3: Scenario I - Warehouse utilization level - 100%
<table>
<thead>
<tr>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>90</td>
<td>22</td>
<td>15</td>
<td>10</td>
<td>21</td>
<td>17</td>
<td>31</td>
<td>90</td>
<td>98</td>
<td>15</td>
<td>56</td>
<td>18</td>
<td>97</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>46</td>
<td>41</td>
<td>45</td>
<td>37</td>
<td>17</td>
<td>32</td>
<td>90</td>
<td>62</td>
<td>25</td>
<td>33</td>
<td>5</td>
<td>61</td>
</tr>
<tr>
<td>3</td>
<td>90</td>
<td>30</td>
<td>9</td>
<td>29</td>
<td>15</td>
<td>24</td>
<td>33</td>
<td>90</td>
<td>56</td>
<td>36</td>
<td>34</td>
<td>55</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>90</td>
<td>40</td>
<td>6</td>
<td>22</td>
<td>20</td>
<td>39</td>
<td>34</td>
<td>90</td>
<td>18</td>
<td>13</td>
<td>12</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>90</td>
<td>56</td>
<td>55</td>
<td>15</td>
<td>33</td>
<td>29</td>
<td>35</td>
<td>90</td>
<td>30</td>
<td>28</td>
<td>29</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>90</td>
<td>47</td>
<td>46</td>
<td>27</td>
<td>42</td>
<td>40</td>
<td>36</td>
<td>90</td>
<td>50</td>
<td>49</td>
<td>46</td>
<td>21</td>
<td>23</td>
</tr>
<tr>
<td>7</td>
<td>90</td>
<td>58</td>
<td>31</td>
<td>22</td>
<td>57</td>
<td>21</td>
<td>37</td>
<td>90</td>
<td>34</td>
<td>33</td>
<td>5</td>
<td>21</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>90</td>
<td>69</td>
<td>30</td>
<td>12</td>
<td>68</td>
<td>20</td>
<td>38</td>
<td>90</td>
<td>35</td>
<td>11</td>
<td>34</td>
<td>33</td>
<td>25</td>
</tr>
<tr>
<td>9</td>
<td>90</td>
<td>44</td>
<td>33</td>
<td>5</td>
<td>4</td>
<td>43</td>
<td>39</td>
<td>90</td>
<td>72</td>
<td>18</td>
<td>18</td>
<td>71</td>
<td>19</td>
</tr>
<tr>
<td>10</td>
<td>90</td>
<td>42</td>
<td>33</td>
<td>24</td>
<td>31</td>
<td>41</td>
<td>40</td>
<td>90</td>
<td>32</td>
<td>31</td>
<td>18</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>90</td>
<td>50</td>
<td>2</td>
<td>49</td>
<td>21</td>
<td>6</td>
<td>41</td>
<td>90</td>
<td>31</td>
<td>10</td>
<td>30</td>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>90</td>
<td>79</td>
<td>16</td>
<td>43</td>
<td>2</td>
<td>78</td>
<td>42</td>
<td>90</td>
<td>26</td>
<td>25</td>
<td>8</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>13</td>
<td>90</td>
<td>56</td>
<td>34</td>
<td>8</td>
<td>47</td>
<td>55</td>
<td>43</td>
<td>90</td>
<td>16</td>
<td>8</td>
<td>15</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>14</td>
<td>90</td>
<td>42</td>
<td>26</td>
<td>41</td>
<td>10</td>
<td>7</td>
<td>44</td>
<td>90</td>
<td>69</td>
<td>19</td>
<td>8</td>
<td>68</td>
<td>43</td>
</tr>
<tr>
<td>15</td>
<td>90</td>
<td>91</td>
<td>90</td>
<td>45</td>
<td>16</td>
<td>41</td>
<td>45</td>
<td>90</td>
<td>44</td>
<td>38</td>
<td>21</td>
<td>43</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>90</td>
<td>80</td>
<td>1</td>
<td>79</td>
<td>17</td>
<td>60</td>
<td>46</td>
<td>90</td>
<td>48</td>
<td>47</td>
<td>30</td>
<td>14</td>
<td>23</td>
</tr>
<tr>
<td>17</td>
<td>90</td>
<td>39</td>
<td>37</td>
<td>25</td>
<td>38</td>
<td>6</td>
<td>47</td>
<td>90</td>
<td>71</td>
<td>25</td>
<td>30</td>
<td>22</td>
<td>70</td>
</tr>
<tr>
<td>18</td>
<td>90</td>
<td>76</td>
<td>19</td>
<td>17</td>
<td>75</td>
<td>18</td>
<td>48</td>
<td>90</td>
<td>31</td>
<td>30</td>
<td>15</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>90</td>
<td>54</td>
<td>32</td>
<td>16</td>
<td>53</td>
<td>21</td>
<td>49</td>
<td>90</td>
<td>43</td>
<td>26</td>
<td>42</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>20</td>
<td>90</td>
<td>58</td>
<td>57</td>
<td>25</td>
<td>20</td>
<td>6</td>
<td>50</td>
<td>90</td>
<td>36</td>
<td>32</td>
<td>35</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>21</td>
<td>90</td>
<td>56</td>
<td>55</td>
<td>36</td>
<td>21</td>
<td>23</td>
<td>51</td>
<td>90</td>
<td>61</td>
<td>33</td>
<td>18</td>
<td>27</td>
<td>60</td>
</tr>
<tr>
<td>22</td>
<td>90</td>
<td>42</td>
<td>1</td>
<td>17</td>
<td>20</td>
<td>41</td>
<td>52</td>
<td>90</td>
<td>85</td>
<td>27</td>
<td>84</td>
<td>57</td>
<td>43</td>
</tr>
<tr>
<td>23</td>
<td>90</td>
<td>53</td>
<td>52</td>
<td>29</td>
<td>1</td>
<td>16</td>
<td>53</td>
<td>90</td>
<td>46</td>
<td>9</td>
<td>14</td>
<td>43</td>
<td>45</td>
</tr>
<tr>
<td>24</td>
<td>90</td>
<td>48</td>
<td>47</td>
<td>38</td>
<td>17</td>
<td>14</td>
<td>54</td>
<td>90</td>
<td>55</td>
<td>37</td>
<td>35</td>
<td>54</td>
<td>40</td>
</tr>
<tr>
<td>25</td>
<td>90</td>
<td>37</td>
<td>34</td>
<td>26</td>
<td>36</td>
<td>28</td>
<td>55</td>
<td>90</td>
<td>47</td>
<td>20</td>
<td>46</td>
<td>36</td>
<td>39</td>
</tr>
<tr>
<td>26</td>
<td>90</td>
<td>54</td>
<td>10</td>
<td>53</td>
<td>26</td>
<td>34</td>
<td>56</td>
<td>90</td>
<td>38</td>
<td>37</td>
<td>18</td>
<td>26</td>
<td>20</td>
</tr>
<tr>
<td>27</td>
<td>90</td>
<td>70</td>
<td>14</td>
<td>5</td>
<td>69</td>
<td>12</td>
<td>57</td>
<td>90</td>
<td>29</td>
<td>28</td>
<td>3</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>28</td>
<td>90</td>
<td>22</td>
<td>7</td>
<td>5</td>
<td>12</td>
<td>21</td>
<td>58</td>
<td>90</td>
<td>47</td>
<td>15</td>
<td>46</td>
<td>38</td>
<td>5</td>
</tr>
<tr>
<td>29</td>
<td>90</td>
<td>70</td>
<td>38</td>
<td>42</td>
<td>25</td>
<td>69</td>
<td>59</td>
<td>90</td>
<td>34</td>
<td>29</td>
<td>33</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>30</td>
<td>90</td>
<td>108</td>
<td>107</td>
<td>62</td>
<td>33</td>
<td>17</td>
<td>60</td>
<td>90</td>
<td>33</td>
<td>30</td>
<td>32</td>
<td>8</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 4: Scenario I - Warehouse utilization level - 90%
<table>
<thead>
<tr>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80</td>
<td>19</td>
<td>16</td>
<td>10</td>
<td>5</td>
<td>18</td>
<td>31</td>
<td>80</td>
<td>27</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td>29</td>
<td>24</td>
<td>8</td>
<td>28</td>
<td>14</td>
<td>32</td>
<td>80</td>
<td>19</td>
<td>13</td>
<td>17</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>28</td>
<td>8</td>
<td>13</td>
<td>16</td>
<td>27</td>
<td>33</td>
<td>80</td>
<td>23</td>
<td>22</td>
<td>12</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td>44</td>
<td>25</td>
<td>43</td>
<td>38</td>
<td>16</td>
<td>35</td>
<td>80</td>
<td>29</td>
<td>21</td>
<td>28</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>80</td>
<td>15</td>
<td>12</td>
<td>14</td>
<td>9</td>
<td>9</td>
<td>36</td>
<td>80</td>
<td>33</td>
<td>18</td>
<td>22</td>
<td>32</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>80</td>
<td>32</td>
<td>17</td>
<td>31</td>
<td>6</td>
<td>20</td>
<td>37</td>
<td>80</td>
<td>19</td>
<td>18</td>
<td>5</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>80</td>
<td>37</td>
<td>36</td>
<td>14</td>
<td>9</td>
<td>4</td>
<td>38</td>
<td>80</td>
<td>61</td>
<td>33</td>
<td>43</td>
<td>13</td>
<td>60</td>
</tr>
<tr>
<td>8</td>
<td>80</td>
<td>32</td>
<td>31</td>
<td>14</td>
<td>11</td>
<td>9</td>
<td>39</td>
<td>80</td>
<td>24</td>
<td>21</td>
<td>23</td>
<td>9</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>80</td>
<td>48</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>47</td>
<td>40</td>
<td>80</td>
<td>26</td>
<td>4</td>
<td>12</td>
<td>7</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>80</td>
<td>18</td>
<td>8</td>
<td>14</td>
<td>16</td>
<td>17</td>
<td>41</td>
<td>80</td>
<td>35</td>
<td>6</td>
<td>34</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>80</td>
<td>30</td>
<td>24</td>
<td>17</td>
<td>11</td>
<td>29</td>
<td>42</td>
<td>80</td>
<td>29</td>
<td>6</td>
<td>21</td>
<td>28</td>
<td>21</td>
</tr>
<tr>
<td>12</td>
<td>80</td>
<td>44</td>
<td>12</td>
<td>28</td>
<td>43</td>
<td>36</td>
<td>43</td>
<td>80</td>
<td>59</td>
<td>44</td>
<td>58</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>13</td>
<td>80</td>
<td>39</td>
<td>19</td>
<td>13</td>
<td>14</td>
<td>38</td>
<td>44</td>
<td>80</td>
<td>41</td>
<td>11</td>
<td>34</td>
<td>17</td>
<td>40</td>
</tr>
<tr>
<td>14</td>
<td>80</td>
<td>40</td>
<td>18</td>
<td>39</td>
<td>21</td>
<td>25</td>
<td>45</td>
<td>80</td>
<td>25</td>
<td>12</td>
<td>8</td>
<td>24</td>
<td>6</td>
</tr>
<tr>
<td>15</td>
<td>80</td>
<td>33</td>
<td>32</td>
<td>11</td>
<td>22</td>
<td>1</td>
<td>46</td>
<td>80</td>
<td>38</td>
<td>1</td>
<td>37</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>16</td>
<td>80</td>
<td>26</td>
<td>25</td>
<td>9</td>
<td>13</td>
<td>22</td>
<td>47</td>
<td>80</td>
<td>33</td>
<td>10</td>
<td>6</td>
<td>28</td>
<td>32</td>
</tr>
<tr>
<td>17</td>
<td>80</td>
<td>26</td>
<td>19</td>
<td>25</td>
<td>12</td>
<td>17</td>
<td>48</td>
<td>80</td>
<td>14</td>
<td>12</td>
<td>5</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>18</td>
<td>80</td>
<td>57</td>
<td>56</td>
<td>17</td>
<td>32</td>
<td>9</td>
<td>49</td>
<td>80</td>
<td>20</td>
<td>2</td>
<td>19</td>
<td>19</td>
<td>7</td>
</tr>
<tr>
<td>19</td>
<td>80</td>
<td>39</td>
<td>13</td>
<td>3</td>
<td>18</td>
<td>38</td>
<td>50</td>
<td>80</td>
<td>20</td>
<td>19</td>
<td>4</td>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>20</td>
<td>80</td>
<td>29</td>
<td>28</td>
<td>13</td>
<td>24</td>
<td>15</td>
<td>51</td>
<td>80</td>
<td>30</td>
<td>12</td>
<td>18</td>
<td>17</td>
<td>29</td>
</tr>
<tr>
<td>21</td>
<td>80</td>
<td>15</td>
<td>6</td>
<td>3</td>
<td>14</td>
<td>5</td>
<td>52</td>
<td>80</td>
<td>29</td>
<td>11</td>
<td>11</td>
<td>28</td>
<td>5</td>
</tr>
<tr>
<td>22</td>
<td>80</td>
<td>44</td>
<td>28</td>
<td>3</td>
<td>7</td>
<td>43</td>
<td>53</td>
<td>80</td>
<td>27</td>
<td>12</td>
<td>23</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>80</td>
<td>28</td>
<td>7</td>
<td>12</td>
<td>14</td>
<td>27</td>
<td>54</td>
<td>80</td>
<td>16</td>
<td>9</td>
<td>14</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>24</td>
<td>80</td>
<td>61</td>
<td>12</td>
<td>60</td>
<td>24</td>
<td>40</td>
<td>55</td>
<td>80</td>
<td>30</td>
<td>29</td>
<td>8</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>25</td>
<td>80</td>
<td>14</td>
<td>13</td>
<td>3</td>
<td>9</td>
<td>6</td>
<td>56</td>
<td>80</td>
<td>19</td>
<td>9</td>
<td>18</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>26</td>
<td>80</td>
<td>41</td>
<td>1</td>
<td>40</td>
<td>16</td>
<td>10</td>
<td>57</td>
<td>80</td>
<td>46</td>
<td>45</td>
<td>24</td>
<td>28</td>
<td>13</td>
</tr>
<tr>
<td>27</td>
<td>80</td>
<td>27</td>
<td>10</td>
<td>9</td>
<td>26</td>
<td>17</td>
<td>58</td>
<td>80</td>
<td>31</td>
<td>15</td>
<td>25</td>
<td>30</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td>80</td>
<td>38</td>
<td>12</td>
<td>6</td>
<td>29</td>
<td>37</td>
<td>59</td>
<td>80</td>
<td>21</td>
<td>20</td>
<td>5</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>29</td>
<td>80</td>
<td>49</td>
<td>27</td>
<td>13</td>
<td>22</td>
<td>48</td>
<td>60</td>
<td>80</td>
<td>40</td>
<td>39</td>
<td>15</td>
<td>27</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 5: Scenario I - Warehouse utilization level - 80%
<table>
<thead>
<tr>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>70</td>
<td>38</td>
<td>37</td>
<td>10</td>
<td>17</td>
<td>21</td>
<td>31</td>
<td>70</td>
<td>32</td>
<td>28</td>
<td>31</td>
<td>16</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>70</td>
<td>30</td>
<td>29</td>
<td>16</td>
<td>17</td>
<td>6</td>
<td>32</td>
<td>70</td>
<td>26</td>
<td>5</td>
<td>17</td>
<td>25</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>70</td>
<td>17</td>
<td>16</td>
<td>14</td>
<td>14</td>
<td>2</td>
<td>33</td>
<td>70</td>
<td>17</td>
<td>6</td>
<td>10</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>70</td>
<td>24</td>
<td>17</td>
<td>11</td>
<td>23</td>
<td>16</td>
<td>34</td>
<td>70</td>
<td>63</td>
<td>2</td>
<td>2</td>
<td>25</td>
<td>62</td>
</tr>
<tr>
<td>5</td>
<td>70</td>
<td>36</td>
<td>15</td>
<td>4</td>
<td>24</td>
<td>35</td>
<td>35</td>
<td>70</td>
<td>16</td>
<td>15</td>
<td>12</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>70</td>
<td>22</td>
<td>21</td>
<td>19</td>
<td>16</td>
<td>3</td>
<td>36</td>
<td>70</td>
<td>31</td>
<td>25</td>
<td>4</td>
<td>19</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>70</td>
<td>23</td>
<td>14</td>
<td>22</td>
<td>20</td>
<td>11</td>
<td>37</td>
<td>70</td>
<td>17</td>
<td>14</td>
<td>16</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>70</td>
<td>27</td>
<td>9</td>
<td>12</td>
<td>5</td>
<td>26</td>
<td>38</td>
<td>70</td>
<td>22</td>
<td>13</td>
<td>21</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>70</td>
<td>25</td>
<td>24</td>
<td>20</td>
<td>6</td>
<td>14</td>
<td>39</td>
<td>70</td>
<td>29</td>
<td>9</td>
<td>14</td>
<td>18</td>
<td>28</td>
</tr>
<tr>
<td>10</td>
<td>70</td>
<td>25</td>
<td>5</td>
<td>3</td>
<td>24</td>
<td>14</td>
<td>40</td>
<td>70</td>
<td>22</td>
<td>16</td>
<td>16</td>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>70</td>
<td>28</td>
<td>27</td>
<td>17</td>
<td>3</td>
<td>22</td>
<td>41</td>
<td>70</td>
<td>17</td>
<td>14</td>
<td>10</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>70</td>
<td>15</td>
<td>8</td>
<td>14</td>
<td>8</td>
<td>5</td>
<td>42</td>
<td>70</td>
<td>18</td>
<td>14</td>
<td>11</td>
<td>17</td>
<td>5</td>
</tr>
<tr>
<td>13</td>
<td>70</td>
<td>17</td>
<td>16</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>43</td>
<td>70</td>
<td>38</td>
<td>37</td>
<td>15</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>14</td>
<td>70</td>
<td>38</td>
<td>8</td>
<td>24</td>
<td>29</td>
<td>37</td>
<td>44</td>
<td>70</td>
<td>22</td>
<td>9</td>
<td>21</td>
<td>13</td>
<td>20</td>
</tr>
<tr>
<td>15</td>
<td>70</td>
<td>23</td>
<td>1</td>
<td>22</td>
<td>3</td>
<td>3</td>
<td>45</td>
<td>70</td>
<td>23</td>
<td>3</td>
<td>19</td>
<td>10</td>
<td>22</td>
</tr>
<tr>
<td>16</td>
<td>70</td>
<td>7</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>46</td>
<td>70</td>
<td>16</td>
<td>15</td>
<td>5</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>17</td>
<td>70</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>47</td>
<td>70</td>
<td>21</td>
<td>2</td>
<td>20</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>18</td>
<td>70</td>
<td>19</td>
<td>5</td>
<td>1</td>
<td>18</td>
<td>5</td>
<td>48</td>
<td>70</td>
<td>20</td>
<td>17</td>
<td>19</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>70</td>
<td>29</td>
<td>18</td>
<td>20</td>
<td>14</td>
<td>28</td>
<td>49</td>
<td>70</td>
<td>22</td>
<td>17</td>
<td>16</td>
<td>21</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>70</td>
<td>15</td>
<td>8</td>
<td>14</td>
<td>13</td>
<td>3</td>
<td>50</td>
<td>70</td>
<td>21</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>70</td>
<td>19</td>
<td>5</td>
<td>18</td>
<td>7</td>
<td>16</td>
<td>51</td>
<td>70</td>
<td>23</td>
<td>16</td>
<td>22</td>
<td>13</td>
<td>18</td>
</tr>
<tr>
<td>22</td>
<td>70</td>
<td>15</td>
<td>14</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>52</td>
<td>70</td>
<td>36</td>
<td>13</td>
<td>3</td>
<td>35</td>
<td>24</td>
</tr>
<tr>
<td>23</td>
<td>70</td>
<td>16</td>
<td>7</td>
<td>12</td>
<td>11</td>
<td>15</td>
<td>53</td>
<td>70</td>
<td>13</td>
<td>9</td>
<td>12</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>24</td>
<td>70</td>
<td>19</td>
<td>14</td>
<td>18</td>
<td>14</td>
<td>2</td>
<td>54</td>
<td>70</td>
<td>47</td>
<td>20</td>
<td>23</td>
<td>18</td>
<td>46</td>
</tr>
<tr>
<td>25</td>
<td>70</td>
<td>18</td>
<td>3</td>
<td>8</td>
<td>8</td>
<td>17</td>
<td>55</td>
<td>70</td>
<td>26</td>
<td>8</td>
<td>25</td>
<td>21</td>
<td>24</td>
</tr>
<tr>
<td>26</td>
<td>70</td>
<td>28</td>
<td>9</td>
<td>27</td>
<td>12</td>
<td>20</td>
<td>56</td>
<td>70</td>
<td>18</td>
<td>13</td>
<td>17</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>27</td>
<td>70</td>
<td>33</td>
<td>32</td>
<td>24</td>
<td>12</td>
<td>3</td>
<td>57</td>
<td>70</td>
<td>17</td>
<td>12</td>
<td>9</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>28</td>
<td>70</td>
<td>18</td>
<td>7</td>
<td>11</td>
<td>17</td>
<td>17</td>
<td>58</td>
<td>70</td>
<td>32</td>
<td>22</td>
<td>9</td>
<td>31</td>
<td>16</td>
</tr>
<tr>
<td>29</td>
<td>70</td>
<td>35</td>
<td>20</td>
<td>34</td>
<td>7</td>
<td>30</td>
<td>59</td>
<td>70</td>
<td>27</td>
<td>8</td>
<td>16</td>
<td>11</td>
<td>26</td>
</tr>
<tr>
<td>30</td>
<td>70</td>
<td>31</td>
<td>11</td>
<td>14</td>
<td>16</td>
<td>30</td>
<td>60</td>
<td>70</td>
<td>24</td>
<td>8</td>
<td>23</td>
<td>12</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 6: Scenario I - Warehouse utilization level - 70%
### Appendix B: Scenario II – Algorithm Performance

<table>
<thead>
<tr>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>52</td>
<td>1</td>
<td>39</td>
<td>2</td>
<td>51</td>
<td>31</td>
<td>100</td>
<td>72</td>
<td>30</td>
<td>27</td>
<td>64</td>
<td>71</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>65</td>
<td>53</td>
<td>53</td>
<td>36</td>
<td>54</td>
<td>32</td>
<td>100</td>
<td>78</td>
<td>44</td>
<td>70</td>
<td>37</td>
<td>77</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>58</td>
<td>55</td>
<td>35</td>
<td>56</td>
<td>45</td>
<td>33</td>
<td>100</td>
<td>53</td>
<td>3</td>
<td>7</td>
<td>52</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>61</td>
<td>59</td>
<td>6</td>
<td>29</td>
<td>7</td>
<td>34</td>
<td>100</td>
<td>48</td>
<td>26</td>
<td>43</td>
<td>47</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>57</td>
<td>3</td>
<td>54</td>
<td>56</td>
<td>55</td>
<td>35</td>
<td>100</td>
<td>73</td>
<td>56</td>
<td>54</td>
<td>34</td>
<td>29</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
<td>73</td>
<td>67</td>
<td>68</td>
<td>35</td>
<td>69</td>
<td>36</td>
<td>100</td>
<td>57</td>
<td>1</td>
<td>14</td>
<td>21</td>
<td>56</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>53</td>
<td>51</td>
<td>15</td>
<td>32</td>
<td>20</td>
<td>37</td>
<td>100</td>
<td>66</td>
<td>9</td>
<td>58</td>
<td>16</td>
<td>33</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>64</td>
<td>15</td>
<td>11</td>
<td>63</td>
<td>50</td>
<td>38</td>
<td>100</td>
<td>76</td>
<td>69</td>
<td>14</td>
<td>54</td>
<td>44</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
<td>83</td>
<td>12</td>
<td>81</td>
<td>31</td>
<td>40</td>
<td>39</td>
<td>100</td>
<td>95</td>
<td>36</td>
<td>73</td>
<td>75</td>
<td>44</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>38</td>
<td>13</td>
<td>36</td>
<td>33</td>
<td>37</td>
<td>40</td>
<td>100</td>
<td>60</td>
<td>39</td>
<td>29</td>
<td>40</td>
<td>59</td>
</tr>
<tr>
<td>11</td>
<td>100</td>
<td>55</td>
<td>11</td>
<td>53</td>
<td>24</td>
<td>54</td>
<td>41</td>
<td>100</td>
<td>66</td>
<td>1</td>
<td>64</td>
<td>12</td>
<td>65</td>
</tr>
<tr>
<td>12</td>
<td>100</td>
<td>50</td>
<td>24</td>
<td>28</td>
<td>25</td>
<td>49</td>
<td>42</td>
<td>100</td>
<td>105</td>
<td>6</td>
<td>79</td>
<td>69</td>
<td>104</td>
</tr>
<tr>
<td>13</td>
<td>100</td>
<td>52</td>
<td>29</td>
<td>13</td>
<td>41</td>
<td>51</td>
<td>43</td>
<td>100</td>
<td>54</td>
<td>51</td>
<td>52</td>
<td>37</td>
<td>53</td>
</tr>
<tr>
<td>14</td>
<td>100</td>
<td>72</td>
<td>30</td>
<td>27</td>
<td>64</td>
<td>71</td>
<td>44</td>
<td>100</td>
<td>117</td>
<td>79</td>
<td>115</td>
<td>63</td>
<td>116</td>
</tr>
<tr>
<td>15</td>
<td>100</td>
<td>78</td>
<td>44</td>
<td>70</td>
<td>37</td>
<td>77</td>
<td>45</td>
<td>100</td>
<td>65</td>
<td>29</td>
<td>19</td>
<td>64</td>
<td>58</td>
</tr>
<tr>
<td>16</td>
<td>100</td>
<td>53</td>
<td>3</td>
<td>7</td>
<td>52</td>
<td>9</td>
<td>46</td>
<td>100</td>
<td>74</td>
<td>62</td>
<td>63</td>
<td>52</td>
<td>66</td>
</tr>
<tr>
<td>17</td>
<td>100</td>
<td>48</td>
<td>26</td>
<td>43</td>
<td>47</td>
<td>41</td>
<td>47</td>
<td>100</td>
<td>62</td>
<td>2</td>
<td>60</td>
<td>26</td>
<td>61</td>
</tr>
<tr>
<td>18</td>
<td>100</td>
<td>73</td>
<td>56</td>
<td>54</td>
<td>34</td>
<td>29</td>
<td>48</td>
<td>100</td>
<td>53</td>
<td>13</td>
<td>51</td>
<td>19</td>
<td>31</td>
</tr>
<tr>
<td>19</td>
<td>100</td>
<td>57</td>
<td>1</td>
<td>14</td>
<td>21</td>
<td>56</td>
<td>49</td>
<td>100</td>
<td>79</td>
<td>77</td>
<td>32</td>
<td>24</td>
<td>51</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>66</td>
<td>9</td>
<td>58</td>
<td>16</td>
<td>33</td>
<td>50</td>
<td>100</td>
<td>67</td>
<td>60</td>
<td>63</td>
<td>65</td>
<td>66</td>
</tr>
<tr>
<td>21</td>
<td>100</td>
<td>76</td>
<td>69</td>
<td>14</td>
<td>54</td>
<td>44</td>
<td>51</td>
<td>100</td>
<td>72</td>
<td>15</td>
<td>47</td>
<td>36</td>
<td>71</td>
</tr>
<tr>
<td>22</td>
<td>100</td>
<td>95</td>
<td>36</td>
<td>73</td>
<td>75</td>
<td>44</td>
<td>52</td>
<td>100</td>
<td>36</td>
<td>14</td>
<td>4</td>
<td>35</td>
<td>16</td>
</tr>
<tr>
<td>23</td>
<td>100</td>
<td>60</td>
<td>39</td>
<td>29</td>
<td>40</td>
<td>59</td>
<td>53</td>
<td>100</td>
<td>89</td>
<td>7</td>
<td>47</td>
<td>88</td>
<td>26</td>
</tr>
<tr>
<td>24</td>
<td>100</td>
<td>66</td>
<td>1</td>
<td>64</td>
<td>12</td>
<td>65</td>
<td>54</td>
<td>100</td>
<td>65</td>
<td>45</td>
<td>51</td>
<td>31</td>
<td>64</td>
</tr>
<tr>
<td>25</td>
<td>100</td>
<td>78</td>
<td>5</td>
<td>45</td>
<td>38</td>
<td>77</td>
<td>55</td>
<td>100</td>
<td>45</td>
<td>43</td>
<td>9</td>
<td>44</td>
<td>42</td>
</tr>
<tr>
<td>26</td>
<td>100</td>
<td>69</td>
<td>20</td>
<td>67</td>
<td>37</td>
<td>31</td>
<td>56</td>
<td>100</td>
<td>49</td>
<td>28</td>
<td>34</td>
<td>5</td>
<td>48</td>
</tr>
<tr>
<td>27</td>
<td>100</td>
<td>56</td>
<td>7</td>
<td>10</td>
<td>12</td>
<td>55</td>
<td>57</td>
<td>100</td>
<td>73</td>
<td>70</td>
<td>55</td>
<td>40</td>
<td>32</td>
</tr>
<tr>
<td>28</td>
<td>100</td>
<td>54</td>
<td>41</td>
<td>42</td>
<td>53</td>
<td>51</td>
<td>58</td>
<td>100</td>
<td>66</td>
<td>64</td>
<td>6</td>
<td>31</td>
<td>20</td>
</tr>
<tr>
<td>29</td>
<td>100</td>
<td>89</td>
<td>23</td>
<td>12</td>
<td>80</td>
<td>88</td>
<td>59</td>
<td>100</td>
<td>82</td>
<td>35</td>
<td>72</td>
<td>65</td>
<td>30</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>77</td>
<td>38</td>
<td>44</td>
<td>75</td>
<td>55</td>
<td>60</td>
<td>100</td>
<td>73</td>
<td>3</td>
<td>56</td>
<td>19</td>
<td>72</td>
</tr>
</tbody>
</table>

Table 7: Scenario II - Warehouse utilization level - 100%
<table>
<thead>
<tr>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>90</td>
<td>38</td>
<td>36</td>
<td>6</td>
<td>37</td>
<td>19</td>
<td>31</td>
<td>90</td>
<td>55</td>
<td>22</td>
<td>47</td>
<td>39</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>37</td>
<td>29</td>
<td>32</td>
<td>35</td>
<td>36</td>
<td>32</td>
<td>90</td>
<td>93</td>
<td>2</td>
<td>74</td>
<td>43</td>
<td>92</td>
</tr>
<tr>
<td>3</td>
<td>90</td>
<td>25</td>
<td>7</td>
<td>2</td>
<td>24</td>
<td>9</td>
<td>33</td>
<td>90</td>
<td>72</td>
<td>48</td>
<td>60</td>
<td>70</td>
<td>71</td>
</tr>
<tr>
<td>4</td>
<td>90</td>
<td>16</td>
<td>4</td>
<td>5</td>
<td>15</td>
<td>8</td>
<td>34</td>
<td>90</td>
<td>35</td>
<td>16</td>
<td>27</td>
<td>25</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>90</td>
<td>29</td>
<td>26</td>
<td>27</td>
<td>12</td>
<td>28</td>
<td>35</td>
<td>90</td>
<td>40</td>
<td>2</td>
<td>30</td>
<td>38</td>
<td>39</td>
</tr>
<tr>
<td>6</td>
<td>90</td>
<td>46</td>
<td>3</td>
<td>5</td>
<td>45</td>
<td>16</td>
<td>36</td>
<td>90</td>
<td>33</td>
<td>7</td>
<td>31</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>90</td>
<td>23</td>
<td>11</td>
<td>21</td>
<td>2</td>
<td>22</td>
<td>37</td>
<td>90</td>
<td>15</td>
<td>9</td>
<td>13</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>90</td>
<td>96</td>
<td>55</td>
<td>51</td>
<td>17</td>
<td>41</td>
<td>38</td>
<td>90</td>
<td>23</td>
<td>2</td>
<td>20</td>
<td>18</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>90</td>
<td>74</td>
<td>70</td>
<td>21</td>
<td>71</td>
<td>73</td>
<td>39</td>
<td>90</td>
<td>35</td>
<td>10</td>
<td>14</td>
<td>5</td>
<td>34</td>
</tr>
<tr>
<td>10</td>
<td>90</td>
<td>37</td>
<td>35</td>
<td>17</td>
<td>4</td>
<td>35</td>
<td>40</td>
<td>90</td>
<td>27</td>
<td>11</td>
<td>25</td>
<td>8</td>
<td>26</td>
</tr>
<tr>
<td>11</td>
<td>90</td>
<td>23</td>
<td>3</td>
<td>6</td>
<td>22</td>
<td>7</td>
<td>41</td>
<td>90</td>
<td>54</td>
<td>28</td>
<td>46</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>90</td>
<td>46</td>
<td>30</td>
<td>34</td>
<td>36</td>
<td>45</td>
<td>42</td>
<td>90</td>
<td>37</td>
<td>11</td>
<td>31</td>
<td>13</td>
<td>17</td>
</tr>
<tr>
<td>13</td>
<td>90</td>
<td>51</td>
<td>47</td>
<td>17</td>
<td>48</td>
<td>30</td>
<td>43</td>
<td>90</td>
<td>39</td>
<td>3</td>
<td>22</td>
<td>26</td>
<td>38</td>
</tr>
<tr>
<td>14</td>
<td>90</td>
<td>67</td>
<td>46</td>
<td>57</td>
<td>11</td>
<td>37</td>
<td>44</td>
<td>90</td>
<td>60</td>
<td>7</td>
<td>58</td>
<td>36</td>
<td>42</td>
</tr>
<tr>
<td>15</td>
<td>90</td>
<td>22</td>
<td>4</td>
<td>6</td>
<td>21</td>
<td>9</td>
<td>45</td>
<td>90</td>
<td>70</td>
<td>23</td>
<td>52</td>
<td>48</td>
<td>46</td>
</tr>
<tr>
<td>16</td>
<td>90</td>
<td>22</td>
<td>20</td>
<td>7</td>
<td>21</td>
<td>6</td>
<td>46</td>
<td>90</td>
<td>44</td>
<td>7</td>
<td>42</td>
<td>27</td>
<td>24</td>
</tr>
<tr>
<td>17</td>
<td>90</td>
<td>78</td>
<td>44</td>
<td>9</td>
<td>22</td>
<td>77</td>
<td>47</td>
<td>90</td>
<td>40</td>
<td>2</td>
<td>32</td>
<td>11</td>
<td>39</td>
</tr>
<tr>
<td>18</td>
<td>90</td>
<td>21</td>
<td>8</td>
<td>19</td>
<td>19</td>
<td>20</td>
<td>48</td>
<td>90</td>
<td>28</td>
<td>25</td>
<td>1</td>
<td>17</td>
<td>14</td>
</tr>
<tr>
<td>19</td>
<td>90</td>
<td>21</td>
<td>6</td>
<td>9</td>
<td>20</td>
<td>19</td>
<td>49</td>
<td>90</td>
<td>41</td>
<td>20</td>
<td>39</td>
<td>27</td>
<td>40</td>
</tr>
<tr>
<td>20</td>
<td>90</td>
<td>33</td>
<td>31</td>
<td>23</td>
<td>32</td>
<td>24</td>
<td>50</td>
<td>90</td>
<td>41</td>
<td>28</td>
<td>2</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>21</td>
<td>90</td>
<td>37</td>
<td>26</td>
<td>3</td>
<td>13</td>
<td>36</td>
<td>51</td>
<td>90</td>
<td>34</td>
<td>31</td>
<td>20</td>
<td>32</td>
<td>33</td>
</tr>
<tr>
<td>22</td>
<td>90</td>
<td>47</td>
<td>31</td>
<td>25</td>
<td>35</td>
<td>44</td>
<td>52</td>
<td>90</td>
<td>29</td>
<td>6</td>
<td>24</td>
<td>27</td>
<td>16</td>
</tr>
<tr>
<td>23</td>
<td>90</td>
<td>52</td>
<td>47</td>
<td>6</td>
<td>48</td>
<td>51</td>
<td>53</td>
<td>90</td>
<td>41</td>
<td>16</td>
<td>34</td>
<td>35</td>
<td>40</td>
</tr>
<tr>
<td>24</td>
<td>90</td>
<td>28</td>
<td>5</td>
<td>24</td>
<td>26</td>
<td>27</td>
<td>54</td>
<td>90</td>
<td>39</td>
<td>32</td>
<td>1</td>
<td>33</td>
<td>38</td>
</tr>
<tr>
<td>25</td>
<td>90</td>
<td>39</td>
<td>34</td>
<td>16</td>
<td>14</td>
<td>19</td>
<td>55</td>
<td>90</td>
<td>56</td>
<td>33</td>
<td>54</td>
<td>34</td>
<td>46</td>
</tr>
<tr>
<td>26</td>
<td>90</td>
<td>25</td>
<td>5</td>
<td>22</td>
<td>13</td>
<td>16</td>
<td>56</td>
<td>90</td>
<td>30</td>
<td>12</td>
<td>16</td>
<td>13</td>
<td>29</td>
</tr>
<tr>
<td>27</td>
<td>90</td>
<td>21</td>
<td>5</td>
<td>19</td>
<td>6</td>
<td>11</td>
<td>57</td>
<td>90</td>
<td>53</td>
<td>16</td>
<td>45</td>
<td>17</td>
<td>26</td>
</tr>
<tr>
<td>28</td>
<td>90</td>
<td>17</td>
<td>6</td>
<td>3</td>
<td>9</td>
<td>16</td>
<td>58</td>
<td>90</td>
<td>28</td>
<td>17</td>
<td>8</td>
<td>18</td>
<td>27</td>
</tr>
<tr>
<td>29</td>
<td>90</td>
<td>20</td>
<td>13</td>
<td>18</td>
<td>9</td>
<td>19</td>
<td>59</td>
<td>90</td>
<td>38</td>
<td>24</td>
<td>32</td>
<td>25</td>
<td>15</td>
</tr>
<tr>
<td>30</td>
<td>90</td>
<td>48</td>
<td>5</td>
<td>38</td>
<td>12</td>
<td>26</td>
<td>60</td>
<td>90</td>
<td>27</td>
<td>25</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 8: Scenario II - Warehouse utilization level - 90%
<table>
<thead>
<tr>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80</td>
<td>24</td>
<td>5</td>
<td>17</td>
<td>20</td>
<td>15</td>
<td>31</td>
<td>80</td>
<td>28</td>
<td>11</td>
<td>6</td>
<td>16</td>
<td>27</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td>39</td>
<td>24</td>
<td>33</td>
<td>8</td>
<td>31</td>
<td>32</td>
<td>80</td>
<td>44</td>
<td>24</td>
<td>2</td>
<td>21</td>
<td>43</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>15</td>
<td>10</td>
<td>3</td>
<td>9</td>
<td>14</td>
<td>33</td>
<td>80</td>
<td>15</td>
<td>8</td>
<td>12</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td>39</td>
<td>21</td>
<td>28</td>
<td>38</td>
<td>29</td>
<td>34</td>
<td>80</td>
<td>40</td>
<td>24</td>
<td>38</td>
<td>15</td>
<td>21</td>
</tr>
<tr>
<td>5</td>
<td>80</td>
<td>22</td>
<td>4</td>
<td>20</td>
<td>3</td>
<td>21</td>
<td>35</td>
<td>80</td>
<td>16</td>
<td>14</td>
<td>4</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>80</td>
<td>34</td>
<td>20</td>
<td>29</td>
<td>32</td>
<td>33</td>
<td>36</td>
<td>80</td>
<td>23</td>
<td>18</td>
<td>19</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>80</td>
<td>19</td>
<td>3</td>
<td>12</td>
<td>18</td>
<td>12</td>
<td>37</td>
<td>80</td>
<td>45</td>
<td>13</td>
<td>35</td>
<td>31</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>80</td>
<td>15</td>
<td>11</td>
<td>3</td>
<td>10</td>
<td>10</td>
<td>38</td>
<td>80</td>
<td>19</td>
<td>17</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>80</td>
<td>31</td>
<td>3</td>
<td>17</td>
<td>30</td>
<td>24</td>
<td>39</td>
<td>80</td>
<td>17</td>
<td>4</td>
<td>15</td>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>80</td>
<td>50</td>
<td>19</td>
<td>48</td>
<td>20</td>
<td>49</td>
<td>40</td>
<td>80</td>
<td>34</td>
<td>16</td>
<td>28</td>
<td>17</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>80</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>8</td>
<td>5</td>
<td>41</td>
<td>80</td>
<td>14</td>
<td>11</td>
<td>6</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>80</td>
<td>35</td>
<td>19</td>
<td>8</td>
<td>26</td>
<td>24</td>
<td>42</td>
<td>80</td>
<td>20</td>
<td>14</td>
<td>12</td>
<td>10</td>
<td>19</td>
</tr>
<tr>
<td>13</td>
<td>80</td>
<td>34</td>
<td>7</td>
<td>32</td>
<td>20</td>
<td>33</td>
<td>43</td>
<td>80</td>
<td>25</td>
<td>23</td>
<td>15</td>
<td>16</td>
<td>11</td>
</tr>
<tr>
<td>14</td>
<td>80</td>
<td>35</td>
<td>25</td>
<td>26</td>
<td>33</td>
<td>34</td>
<td>44</td>
<td>80</td>
<td>12</td>
<td>3</td>
<td>10</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>15</td>
<td>80</td>
<td>22</td>
<td>16</td>
<td>5</td>
<td>14</td>
<td>21</td>
<td>45</td>
<td>80</td>
<td>59</td>
<td>11</td>
<td>41</td>
<td>7</td>
<td>58</td>
</tr>
<tr>
<td>16</td>
<td>80</td>
<td>34</td>
<td>31</td>
<td>32</td>
<td>27</td>
<td>23</td>
<td>46</td>
<td>80</td>
<td>11</td>
<td>9</td>
<td>8</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>17</td>
<td>80</td>
<td>17</td>
<td>14</td>
<td>10</td>
<td>7</td>
<td>16</td>
<td>47</td>
<td>80</td>
<td>11</td>
<td>6</td>
<td>9</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>18</td>
<td>80</td>
<td>31</td>
<td>19</td>
<td>20</td>
<td>27</td>
<td>30</td>
<td>48</td>
<td>80</td>
<td>28</td>
<td>25</td>
<td>3</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td>80</td>
<td>17</td>
<td>8</td>
<td>15</td>
<td>9</td>
<td>11</td>
<td>49</td>
<td>80</td>
<td>37</td>
<td>35</td>
<td>5</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>80</td>
<td>26</td>
<td>24</td>
<td>3</td>
<td>21</td>
<td>8</td>
<td>50</td>
<td>80</td>
<td>10</td>
<td>5</td>
<td>2</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>21</td>
<td>80</td>
<td>39</td>
<td>15</td>
<td>37</td>
<td>20</td>
<td>14</td>
<td>51</td>
<td>80</td>
<td>22</td>
<td>20</td>
<td>5</td>
<td>21</td>
<td>17</td>
</tr>
<tr>
<td>22</td>
<td>80</td>
<td>19</td>
<td>4</td>
<td>14</td>
<td>17</td>
<td>18</td>
<td>52</td>
<td>80</td>
<td>18</td>
<td>3</td>
<td>1</td>
<td>14</td>
<td>17</td>
</tr>
<tr>
<td>23</td>
<td>80</td>
<td>18</td>
<td>5</td>
<td>15</td>
<td>9</td>
<td>9</td>
<td>53</td>
<td>80</td>
<td>30</td>
<td>14</td>
<td>24</td>
<td>13</td>
<td>29</td>
</tr>
<tr>
<td>24</td>
<td>80</td>
<td>30</td>
<td>28</td>
<td>21</td>
<td>29</td>
<td>22</td>
<td>54</td>
<td>80</td>
<td>10</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>25</td>
<td>80</td>
<td>19</td>
<td>17</td>
<td>15</td>
<td>7</td>
<td>10</td>
<td>55</td>
<td>80</td>
<td>27</td>
<td>18</td>
<td>4</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td>26</td>
<td>80</td>
<td>20</td>
<td>17</td>
<td>11</td>
<td>18</td>
<td>19</td>
<td>56</td>
<td>80</td>
<td>24</td>
<td>21</td>
<td>16</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>27</td>
<td>80</td>
<td>31</td>
<td>2</td>
<td>24</td>
<td>6</td>
<td>30</td>
<td>57</td>
<td>80</td>
<td>26</td>
<td>7</td>
<td>24</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>28</td>
<td>80</td>
<td>11</td>
<td>9</td>
<td>4</td>
<td>10</td>
<td>6</td>
<td>58</td>
<td>80</td>
<td>35</td>
<td>18</td>
<td>33</td>
<td>10</td>
<td>22</td>
</tr>
<tr>
<td>29</td>
<td>80</td>
<td>28</td>
<td>4</td>
<td>5</td>
<td>27</td>
<td>26</td>
<td>59</td>
<td>80</td>
<td>27</td>
<td>24</td>
<td>18</td>
<td>14</td>
<td>26</td>
</tr>
<tr>
<td>30</td>
<td>80</td>
<td>18</td>
<td>16</td>
<td>2</td>
<td>17</td>
<td>8</td>
<td>60</td>
<td>80</td>
<td>14</td>
<td>10</td>
<td>3</td>
<td>13</td>
<td>7</td>
</tr>
</tbody>
</table>

**Table 9: Scenario II - Warehouse utilization level - 80%**
<table>
<thead>
<tr>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
<th>Iteration Number</th>
<th>Density</th>
<th>Retrieval Time</th>
<th>H1 Time</th>
<th>H2 Time</th>
<th>H3 Time</th>
<th>H4 Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>70</td>
<td>28</td>
<td>26</td>
<td>20</td>
<td>27</td>
<td>21</td>
<td>31</td>
<td>70</td>
<td>28</td>
<td>23</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>70</td>
<td>16</td>
<td>14</td>
<td>1</td>
<td>15</td>
<td>6</td>
<td>32</td>
<td>70</td>
<td>35</td>
<td>27</td>
<td>18</td>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>3</td>
<td>70</td>
<td>30</td>
<td>7</td>
<td>28</td>
<td>7</td>
<td>16</td>
<td>33</td>
<td>70</td>
<td>16</td>
<td>7</td>
<td>14</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>70</td>
<td>21</td>
<td>12</td>
<td>6</td>
<td>12</td>
<td>20</td>
<td>34</td>
<td>70</td>
<td>11</td>
<td>2</td>
<td>3</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>70</td>
<td>18</td>
<td>11</td>
<td>14</td>
<td>9</td>
<td>15</td>
<td>35</td>
<td>70</td>
<td>23</td>
<td>20</td>
<td>4</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>70</td>
<td>18</td>
<td>16</td>
<td>6</td>
<td>17</td>
<td>8</td>
<td>36</td>
<td>70</td>
<td>36</td>
<td>9</td>
<td>34</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>70</td>
<td>29</td>
<td>20</td>
<td>27</td>
<td>6</td>
<td>28</td>
<td>37</td>
<td>70</td>
<td>21</td>
<td>4</td>
<td>16</td>
<td>17</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>70</td>
<td>14</td>
<td>9</td>
<td>9</td>
<td>12</td>
<td>13</td>
<td>38</td>
<td>70</td>
<td>30</td>
<td>20</td>
<td>7</td>
<td>6</td>
<td>29</td>
</tr>
<tr>
<td>9</td>
<td>70</td>
<td>23</td>
<td>7</td>
<td>21</td>
<td>8</td>
<td>22</td>
<td>39</td>
<td>70</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>70</td>
<td>17</td>
<td>12</td>
<td>7</td>
<td>12</td>
<td>16</td>
<td>40</td>
<td>70</td>
<td>22</td>
<td>14</td>
<td>7</td>
<td>8</td>
<td>21</td>
</tr>
<tr>
<td>11</td>
<td>70</td>
<td>25</td>
<td>9</td>
<td>23</td>
<td>9</td>
<td>24</td>
<td>41</td>
<td>70</td>
<td>13</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>12</td>
<td>70</td>
<td>12</td>
<td>7</td>
<td>10</td>
<td>8</td>
<td>11</td>
<td>42</td>
<td>70</td>
<td>13</td>
<td>2</td>
<td>11</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>70</td>
<td>23</td>
<td>21</td>
<td>5</td>
<td>13</td>
<td>18</td>
<td>43</td>
<td>70</td>
<td>18</td>
<td>8</td>
<td>16</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>14</td>
<td>70</td>
<td>24</td>
<td>9</td>
<td>20</td>
<td>8</td>
<td>20</td>
<td>44</td>
<td>70</td>
<td>20</td>
<td>8</td>
<td>17</td>
<td>9</td>
<td>18</td>
</tr>
<tr>
<td>15</td>
<td>70</td>
<td>16</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>15</td>
<td>45</td>
<td>70</td>
<td>27</td>
<td>11</td>
<td>25</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>70</td>
<td>18</td>
<td>15</td>
<td>3</td>
<td>12</td>
<td>4</td>
<td>46</td>
<td>70</td>
<td>15</td>
<td>9</td>
<td>13</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>17</td>
<td>70</td>
<td>28</td>
<td>7</td>
<td>23</td>
<td>26</td>
<td>10</td>
<td>47</td>
<td>70</td>
<td>13</td>
<td>3</td>
<td>11</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>18</td>
<td>70</td>
<td>23</td>
<td>3</td>
<td>16</td>
<td>17</td>
<td>22</td>
<td>48</td>
<td>70</td>
<td>13</td>
<td>9</td>
<td>1</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td>70</td>
<td>23</td>
<td>8</td>
<td>5</td>
<td>22</td>
<td>17</td>
<td>49</td>
<td>70</td>
<td>23</td>
<td>19</td>
<td>9</td>
<td>21</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>70</td>
<td>20</td>
<td>12</td>
<td>5</td>
<td>18</td>
<td>19</td>
<td>50</td>
<td>70</td>
<td>16</td>
<td>6</td>
<td>2</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>21</td>
<td>70</td>
<td>21</td>
<td>9</td>
<td>19</td>
<td>11</td>
<td>20</td>
<td>51</td>
<td>70</td>
<td>19</td>
<td>6</td>
<td>14</td>
<td>18</td>
<td>15</td>
</tr>
<tr>
<td>22</td>
<td>70</td>
<td>30</td>
<td>5</td>
<td>23</td>
<td>9</td>
<td>29</td>
<td>52</td>
<td>70</td>
<td>21</td>
<td>5</td>
<td>18</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>23</td>
<td>70</td>
<td>25</td>
<td>12</td>
<td>17</td>
<td>11</td>
<td>9</td>
<td>53</td>
<td>70</td>
<td>15</td>
<td>8</td>
<td>13</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>24</td>
<td>70</td>
<td>19</td>
<td>10</td>
<td>2</td>
<td>12</td>
<td>18</td>
<td>54</td>
<td>70</td>
<td>26</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>25</td>
</tr>
<tr>
<td>25</td>
<td>70</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>55</td>
<td>70</td>
<td>23</td>
<td>14</td>
<td>21</td>
<td>11</td>
<td>22</td>
</tr>
<tr>
<td>26</td>
<td>70</td>
<td>14</td>
<td>3</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>56</td>
<td>70</td>
<td>24</td>
<td>18</td>
<td>22</td>
<td>21</td>
<td>17</td>
</tr>
<tr>
<td>27</td>
<td>70</td>
<td>19</td>
<td>4</td>
<td>16</td>
<td>1</td>
<td>9</td>
<td>57</td>
<td>70</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>28</td>
<td>70</td>
<td>20</td>
<td>17</td>
<td>4</td>
<td>12</td>
<td>19</td>
<td>58</td>
<td>70</td>
<td>21</td>
<td>13</td>
<td>14</td>
<td>17</td>
<td>20</td>
</tr>
<tr>
<td>29</td>
<td>70</td>
<td>25</td>
<td>15</td>
<td>11</td>
<td>8</td>
<td>18</td>
<td>59</td>
<td>70</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>30</td>
<td>70</td>
<td>39</td>
<td>19</td>
<td>6</td>
<td>8</td>
<td>38</td>
<td>60</td>
<td>70</td>
<td>34</td>
<td>14</td>
<td>27</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 10 : Scenario II - Warehouse utilization level - 70%
Appendix C: Java code

/* Initiating Program */
package priorityBasedMovement;
import java.io.IOException;
public class Startup {
    public static int length;
    public static int width;
    public static int density;
    public static void main(String[] args) throws IOException{
        String warehouseLength = "8";
        length = Integer.parseInt(warehouseLength);
        String warehouseWidth = "8";
        width = Integer.parseInt(warehouseWidth);
        String utilizationPercentage = "85";
        density = Integer.parseInt(utilizationPercentage);
        Layout layout = new Layout();
        layout.setMap();
    }
}

/* Defining a module and its attributes */
package priorityBasedMovement;
import java.awt.Color;
import java.util.*;
public class Module {
    private int priority;
    private int state;
    private int location;
    private int rowid;
    private int columnid;
    private int inheritedPriority=0;
    private int selectionVariable =0;
    private int destination;
    private int destinationTemp;
    private int destinationTempVariable=0;
    private int nextLocation;
    private int rightLocation;
    private int leftLocation;
    private int bottomLocation;
    private int topLocation;
    private int orderVariable;
    private int orderListConflictVariable=0;
    private int orderListLoadTimeStepCount=0;
    private ArrayList<Integer> sequenceOfTaggedModules = new ArrayList<Integer>();
    private static ArrayList<Integer> locationsForPriorities = new ArrayList<Integer>();
    final int locations=Startup.length*Startup.width;
    private Color color;
    public Module(){
    }
    public Module(int locations,int rowid,int columnid){
        setLocation(locations);
        setState(locations,rowid,columnid);
        setPriority(rowid,columnid);
    }
}
setRowId(rowid);
setColumnId(columnId);
setColor();
setInheritedPriority(0);
setSelectionVariable();
setRightLocation();
setLeftLocation();
setTopLocation();
setBottomLocation();
setOrderVariable(0);
}
public int getLocation() {
    return location;
}
public void setLocation(int locations) {
    location = locations;
}
public int getState() {
    return state;
}
public void setState(int locations, int rowid, int columnId) {
    Random generator = new Random();
    state = (generator.nextInt(100) + 1 < Startup.density) ? 1 : 0;
}
public void setState(int state) {
    this.state = state;
}
public int getPriority() {
    return priority;
}
public void setPriority(int rowid, int columnId) {
    if (this.state == 1)
    {
        Random generator = new Random();
        int tempPriority = generator.nextInt(locations) + 1;
        while (locationsForPriorities.contains(tempPriority)) {
            tempPriority = generator.nextInt(locations) + 1;
        }
        priority = tempPriority;
        locationsForPriorities.add(tempPriority);
    }
    else {
        priority = 0;
    }
}
public void setPriority(int priority) {
    this.priority = priority;
}
public int getRowId() {
    return rowid;
}
public void setRowId(int rowid) {
    this.rowId = rowid;
}
public int getColumnId(){
    return columnid;
}
public void setColumnId(int columnid){
    this.columnid = columnid;
}
public int getInheritedPriority(){
    return inheritedPriority;
}
public void setInheritedPriority(int inheritedPriority){
    this.inheritedPriority = inheritedPriority;
}
public Color getColor(){
    return color;
}
public void setColor(){
    RandomColorGenerator randomColor = new RandomColorGenerator();
    color = randomColor.randomColor().darker();
}
public void setColor(Color color){
    this.color = color;
}
public int getSelectionVariable(){
    return selectionVariable;
}
public void setSelectionVariable(){
    selectionVariable = 0;
}
public void setSelectionVariable(int selectionVariable){
    this.selectionVariable = selectionVariable;
}
public int getDestination(){
    return destination;
}
public void setDestination(int destination){
    this.destination = destination;
}
public int getNextLocation(){
    return nextLocation;
}
public void setNextLocation(int nextLocation){
    this.nextLocation = nextLocation;
}
public int getRightLocation(){
    if (columnid == Startup.width - 1){
        rightLocation = 0;
    } else{
        rightLocation = location + 1;
    }
}
public int getLeftLocation() {
    return leftLocation;
}
public void setLeftLocation() {
    if(columnid == 0){
        leftLocation = 0;
    } else{
        leftLocation = location - 1;
    }
}
public int getBottomLocation() {
    return bottomLocation;
}
public void setBottomLocation() {
    if(rowid == Startup.length-1){
        bottomLocation = 0;
    } else{
        bottomLocation = location + Startup.width;
    }
}
public int getTopLocation() {
    return topLocation;
}
public void setTopLocation() {
    if(rowid == 0){
        topLocation = 0;
    } else{
        topLocation = location - Startup.width;
    }
}
public int getOrderVariable() {
    return orderVariable;
}
public void setOrderVariable(int orderVariable) {
    this.orderVariable = orderVariable;
}
public ArrayList<Integer> getSequenceOfTaggedModules() {
    return sequenceOfTaggedModules;
}
public void setSequenceOfTaggedModules(ArrayList<Integer> pathToNextLocation) {
    this.sequenceOfTaggedModules = pathToNextLocation;
}
public int getOrderListConflictVariable() {
    return orderListConflictVariable;
}
public void setOrderListConflictVariable(int orderListConflictVariable) {
    this.orderListConflictVariable = orderListConflictVariable;
}
public void setOrderListLoadTimeStepCount(int orderListLoadTimeStepCount) {
    this.orderListLoadTimeStepCount = orderListLoadTimeStepCount;
}
public int getOrderListLoadTimeStepCount() {
    return orderListLoadTimeStepCount;
}

public void setDestinationTemp(int destinationTemp) {
    this.destinationTemp = destinationTemp;
}

public int getDestinationTemp() {
    return destinationTemp;
}

public void setDestinationTempVariable(int destinationTempVariable) {
    this.destinationTempVariable = destinationTempVariable;
}

public int getDestinationTempVariable() {
    return destinationTempVariable;
}

import java.awt.Color;
import java.util.*;

public class Layout extends Module{

    public Layout() {
        super();
    }

    public Layout(int count, int rowid, int columnid) {
        super(count, rowid, columnid);
    }

    static Layout[][] layout;
    HashMap<Integer,Layout> map = new HashMap<Integer,Layout>();
    WarehouseMap warehouseMap;
    Set<Integer> blockedLocations = new TreeSet<Integer>();
    List<Integer> orderList;
    ArrayList<Integer> listOfModulesAtDestination = new ArrayList<Integer>();
    WarehouseMap wMap;
    static ArrayList<Integer> docks = new ArrayList<Integer>(Arrays.asList(4,5,12,13));
    int loadsAtDestination=0;
    int numberOfTimeSteps=1;
    static int maxNumberOfTimeStepsForAlgorithm = 300;

    public void setMap(){
        // Setting warehouse layout map
        int count =0;
        layout = new Layout[Startup.length][Startup.width];
        for (int i = 0;i<Startup.length;i++){
            for (int j=0;j<Startup.width;j++){
                count = count + 1;
                layout[i][j] = new Layout(count,i,j);
                map.put(layout[i][j].getLocation(), layout[i][j]);
                if(layout[i][j].getPriority()==0){
                    layout[i][j].setColor(Color.WHITE);
                }
            }
        }
        setDockStations();
        setOrder();
    }
}
warehouseMap = new WarehouseMap();
displayMap(0);
controlLoadMovement();
}
public void setDockStations(){
// Assigning Dock Stations - Can be changed to shift location of the dock
for (int i = 0; i<Startup.length; i++){
    for (int j=0; j<Startup.width; j++){
        if(docks.contains(layout[i][j].getLocation())){
            layout[i][j].setColor(Color.white);
            layout[i][j].setInheritedPriority(0);
            layout[i][j].setPriority(0);
            layout[i][j].setState(0);
            layout[i][j].setOrderVariable(0);
        }
    }
}
public void setOrder(){
    ArrayList<Integer> packagePriorityList = getPackagePriorityList();
    orderList = packagePriorityList.subList(0, 4);
    for(int i=0; i<orderList.size(); i++){
        map.get(orderList.get(i)).setColor(Color.yellow.darker());
        map.get(orderList.get(i)).setDestination(docks.get(i));
        map.get(orderList.get(i)).setInheritedPriority(map.get(orderList.get(i)).getPriority());
        map.get(orderList.get(i)).setOrderVariable(1);
        map.get(orderList.get(i)).setSelectVariable(2);
    }
}
public ArrayList<Integer> getPackagePriorityList(){
    // Sorting packages by priority
    Comparator<Integer> reverse = Collections.reverseOrder();
    TreeMap<Integer, Integer> mapPriority = new TreeMap<Integer, Integer>(reverse);
    for (int i = 0; i<Startup.length; i++){
        for (int j=0; j<Startup.width; j++){
            if(!(layout[i][j].getPriority() == 0)){
                mapPriority.put(layout[i][j].getPriority(), layout[i][j].getLocation());
            }
        }
    }
    ArrayList<Integer> mapPriorityValues = new ArrayList<Integer>(mapPriority.values());
    return mapPriorityValues;
}
public void controlLoadMovement(){
    int flag=0;
    while(numberOfTimeSteps < maxNumberOfTimeStepsForAlgorithm && flag==0){
        // Checking for loads at Destinations
        for(int i = 0; i<orderList.size(); i++){
            if(map.get(orderList.get(i)).getLocation()== map.get(orderList.get(i)).getDestination()&&
            map.get(orderList.get(i)).getSelectVariable() != 3){
                if(map.get(orderList.get(i)).getDestinationTempVariable()==1){
                    map.get(orderList.get(i)).setDestination(map.get(orderList.get(i)).getDestinationTemp());
                    map.get(orderList.get(i)).setDestinationTempVariable(0);
                }
            }
        }
    }
}
else{
    listOfModulesAtDestination.add(map.get(orderList.get(i)).getLocation());
    map.get(orderList.get(i)).setSelectionVariable(3);
    ArrayList<Integer> loadAtDestination = new ArrayList<Integer>(Arrays.asList(map.get(orderList.get(i)).getLocation()));
    updateLoadVariables(3, loadAtDestination);
}

flag=allModulesAtDestination();
// Processing Order Loads
for(int j =0; j<orderList.size(); j++){
    if(map.get(orderList.get(j)).getLocation() != map.get(orderList.get(j)).getDestination()){
        if(map.get(orderList.get(j)).getOrderListConflictVariable()==0){
            ArrayList<Integer> orderModuleForResetting = new ArrayList<Integer>();
            orderModuleForResetting.add(map.get(orderList.get(j)).getLocation());
            updateLoadVariables(4, orderModuleForResetting);
            ArrayList<Integer> sequenceOfTaggedModules = new ArrayList<Integer>();
            if(map.get(orderList.get(j)).getSequenceOfTaggedModules().isEmpty()){ // Setting next location by calling local search
                map.get(orderList.get(j)).setNextLocation(localSearchForMovement(map.get(orderList.get(j))));
                if(map.get(orderList.get(j)).getNextLocation()!=""){
                    if(map.get(orderList.get(j)).getNextLocation()).getPriority()==0){
                        // Tagging Modules
                        sequenceOfTaggedModules.add(map.get(orderList.get(j)).getNextLocation());
                        sequenceOfTaggedModules.add(map.get(orderList.get(j)).getNextLocation());
                    }
                    // Checking for Order Conflict
                    for(int m =0; m<orderList.size(); m++){
                        if(map.get(orderList.get(j)).getSequenceOfTaggedModules().contains(orderList.get(m)) && orderList.get(j) != orderList.get(m)){
                            map.get(orderList.get(m)).setOrderListConflictVariable(1);
                        }
                    }
                }
            }
        }
    }
}
}
else{
    // Setting Temporary Destination
    if(otherLoadsAtDestination(orderList.get(j))){
        setTemporaryDestinationForLoad(orderList.get(j));
    }
}
}
} else{
    resetLoadVariables(2,map.get(orderList.get(j)).getSequenceOfTaggedModules());
    ArrayList<Integer> emptyList = new ArrayList<Integer>();
    map.get(orderList.get(j)).setSequenceOfTaggedModules(emptyList);
}
} else{
    resetLoadVariables(2,map.get(orderList.get(j)).getSequenceOfTaggedModules());
    ArrayList<Integer> emptyList = new ArrayList<Integer>();
    map.get(orderList.get(j)).setSequenceOfTaggedModules(emptyList);
}
}
// Movement of Loads
for(int k =0; k<orderList.size(); k++){
    if(map.get(orderList.get(k)).getLocation() != map.get(orderList.get(k)).getDestination()){
        if(map.get(orderList.get(k)).getOrderListConflictVariable() == 0){
            if(map.get(orderList.get(k)).getNextLocation()!=0){
                ArrayList<Integer> pathToNextLocation = map.get(orderList.get(k)).getSequenceOfTaggedModules();
                if(!pathToNextLocation.isEmpty()){  
                    getMovementForCurrentTimeStep(pathToNextLocation,k);
                }
            }
            map.get(orderList.get(k)).setOrderListLoadTimeStepCount(map.get(orderList.get(k)).getOrderListLoadTimeStepCount()+1);
        }
        else{
            resetLoadVariables(2,map.get(orderList.get(k)).getSequenceOfTaggedModules());
            ArrayList<Integer> emptyList = new ArrayList<Integer>();
            map.get(orderList.get(k)).setSequenceOfTaggedModules(emptyList);
            map.get(orderList.get(k)).setOrderListLoadTimeStepCount(map.get(orderList.get(k)).getOrderListLoadTimeStepCount()+1);
        }
    }
    else{
        resetLoadVariables(2,map.get(orderList.get(k)).getSequenceOfTaggedModules());
        ArrayList<Integer> emptyList = new ArrayList<Integer>();
        map.get(orderList.get(k)).setSequenceOfTaggedModules(emptyList);
    }
}
displayMap(500);
numberOfTimeSteps++;
// Checking for other higher priority loads at destination
public boolean otherLoadsAtDestination(int currentOrderModule)
    {
        int count=0;
        boolean result;
        for(int i=0;i<orderList.size();i++)
            {
                if(currentOrderModule != orderList.get(i) && map.get(currentOrderModule).getPriority() < map.get(orderList.get(i)).getPriority()){
                    if(map.get(orderList.get(i)).getLocation()== map.get(orderList.get(i)).getDestination()){
                        count++;
                    }
                    else{
                        break;
                    }
                }
            }
        if(count == (orderList.indexOf(currentOrderModule))){
            result=true;
        }
        else {
            result=false;
        }
        return result;
    }
// Setting Temporary destination
public void setTemporaryDestinationForLoad(int currentOrderModule){
    int tempDest=0;
    for (int i = 0;i<Startup.length;i++)
        {
            for (int j=0;j<Startup.width;j++)
                {
                    if(layout[i][j].getLocation()==currentOrderModule && map.get(currentOrderModule).getDestinationTemp()==0){
                        if(i==0){
                            tempDest = map.get(currentOrderModule).getDestination()+2*Startup.width;
                        }
                        else{
                            tempDest = map.get(currentOrderModule).getDestination()+Startup.width;
                        }
                        map.get(currentOrderModule).setDestinationTemp(map.get(currentOrderModule).getDestination());
                        map.get(currentOrderModule).setDestination(tempDest);
                        map.get(currentOrderModule).setDestinationTempVariable(1);
                    }
                }
        }
    }
// Block Movement for Order Loads
public void getMovementForCurrentTimeStep(ArrayList<Integer> loadPathToNextLocation, int orderListIteratorPosition){
    Collections.reverse(loadPathToNextLocation);
    if(loadPathToNextLocation.size()==2){
        moveLoadsToEmptySpace(loadPathToNextLocation);
        resetLoadVariables(1,loadPathToNextLocation);
        loadPathToNextLocation.remove(1);
        loadPathToNextLocation.remove(0);
        map.get(orderList.get(orderListIteratorPosition)).setSequenceOfTaggedModules(loadPathToNextLocation);
else if (loadPathToNextLocation.size()>2){
    ArrayList<Integer> pathForMotion = new ArrayList<Integer>();
    ArrayList<Integer> pathForNextIterations = new ArrayList<Integer>();
    int i=0;
    int count =0;
    while(i<loadPathToNextLocation.size()-2){
        if(loadPathToNextLocation.get(i+1)-loadPathToNextLocation.get(i) == loadPathToNextLocation.get(i+2)-loadPathToNextLocation.get(i+1)){
            count = count+1;
        }
        else{
            break;
        }
        i++;
    }
    for(int j=0;j<count+2;j++){
        pathForMotion.add(loadPathToNextLocation.get(j));
    }
    moveLoadsToEmptySpace(pathForMotion);
    Collections.reverse(pathForMotion);
    if(count != loadPathToNextLocation.size()-2){
        for(int k=count+1;k<loadPathToNextLocation.size();k++){
            pathForNextIterations.add(loadPathToNextLocation.get(k));
        }
        Collections.reverse(pathForNextIterations);
    }
    resetLoadVariables(1,pathForMotion);
    updateLoadVariables(2,pathForNextIterations);
    map.get(orderList.get(orderListIteratorPosition)).setSequenceOfTaggedModules(pathForNextIterations);
}

// Checking for all modules for destination
public int allModulesAtDestination(){
    int flag=0;
    int count =0;
    for(int i=0;i<orderList.size();i++){
        if(map.get(orderList.get(i)).getLocation() == map.get(orderList.get(i)).getDestination()){
            count = count+1;
        }
    }
    loadsAtDestination=count;
    if(count==orderList.size()){
        flag=1;
    }
    return flag;
}
// Update load variables
public void updateLoadVariables(int caseNumber, List<Integer> listOfModules){
    switch(caseNumber){
        case 1:
            // Updating OrderLists
            if(orderList.contains(listOfModules.get(1))){
                int index = orderList.indexOf(listOfModules.get(1));
            }
        }
    }
orderList.set(index, listOfModules.get(0));
}
else if(orderList.contains(listOfModules.get(0))){
    int index = orderList.indexOf(listOfModules.get(0));
    orderList.set(index, listOfModules.get(1));
}
break;
// Updating path for next iteration

case 2:
    for(int j=0; j<listOfModules.size(); j++){
        // Resetting Selection Variables
        if(!orderList.contains(listOfModules.get(j))){
            map.get(listOfModules.get(j)).setSelectionVariable(1);
        }
    }
    break;

// Clearing Inheritance

case 3:
    for (int i = 0; i<Startup.length; i++){
        for (int j=0; j<Startup.width; j++){

            if(map.get(layout[i][j].getLocation()).getInheritedPriority()==map.get(listOfModules.get(0)).getPriority()
                & & map.get(layout[i][j].getLocation()).getSelectionVariable() != 3
                & & map.get(layout[i][j].getLocation()).getSelectionVariable() != 2
                & & map.get(layout[i][j].getLocation()).getLocation() != map.get(listOfModules.get(0)).getLocation()){)
                map.get(layout[i][j].getLocation()).setInheritedPriority(0);
                map.get(layout[i][j].getLocation()).setSelectionVariable(0);
            }
        }
    }
    break;

// Updating Order Lists

case 4:
    for(int j=0; j<listOfModules.size(); j++){
        map.get(listOfModules.get(j)).setInheritedPriority(map.get(listOfModules.get(j)).getPriority());
        map.get(listOfModules.get(j)).setOrderVariable(1);
        map.get(listOfModules.get(j)).setSelectionVariable(2);
    }
    break;

// Updating Order List

case 5:
    ArrayList<Integer> allPaths = new ArrayList<Integer>();
    for(int j=0; j<orderList.size(); j++){
        allPaths.addAll(map.get(orderList.get(j)).getSequenceOfTaggedModules());
    }
    if(!allPaths.contains(listOfModules.get(0))){
        map.get(listOfModules.get(0)).setInheritedPriority(map.get(listOfModules.get(0)).getPriority());
        map.get(listOfModules.get(0)).setOrderVariable(1);
        map.get(listOfModules.get(0)).setSelectionVariable(2);
        map.get(listOfModules.get(0)).setOrderListConflictVariable(0);
    }
}
allPaths.clear();
break;
}

public void resetLoadVariables(int caseNumber, ArrayList<Integer> loadsForResettingVariables){
    switch(caseNumber){
    // resetting loads
    case 1:
        for(int j=0;j<loadsForResettingVariables.size();j++){
            if(!orderList.contains(loadsForResettingVariables.get(j))){
                map.get(loadsForResettingVariables.get(j)).setInheritedPriority(0);
                map.get(loadsForResettingVariables.get(j)).setSelectionVariable(0);
                if(blockedLocations.contains(loadsForResettingVariables.get(j))){
                    blockedLocations.remove(loadsForResettingVariables.get(j));
                }
            }
            else{
                map.get(loadsForResettingVariables.get(j)).setOrderListConflictVariable(0);
                map.get(loadsForResettingVariables.get(j)).setInheritedPriority(map.get(loadsForResettingVariables.get(j)).getPriority());
                map.get(loadsForResettingVariables.get(j)).setSelectionVariable(2);
            }
        }
        break;
    case 2:
        for(int j=0;j<loadsForResettingVariables.size();j++){
            if(!orderList.contains(loadsForResettingVariables.get(j))){
                map.get(loadsForResettingVariables.get(j)).setInheritedPriority(0);
                map.get(loadsForResettingVariables.get(j)).setSelectionVariable(0);
                if(blockedLocations.contains(loadsForResettingVariables.get(j))){
                    blockedLocations.remove(loadsForResettingVariables.get(j));
                }
            }
        }
        break;
    }
    // Returns distance
    public int calculateDistanceBetweenLoads(int firstLocation, int secondLocation){
        int distance=0;
        int locationOneRowId = 0;
        int locationTwoRowId = 0;
        int locationOneColId = 0;
        int locationTwoColId = 0;
        locationOneRowId = map.get(firstLocation).getRowId();
        locationTwoRowId = map.get(secondLocation).getRowId();
        locationOneColId = map.get(firstLocation).getColumnId();
        locationTwoColId = map.get(secondLocation).getColumnId();
        distance = Math.abs(locationOneRowId-locationTwoRowId)+ Math.abs(locationOneColId-locationTwoColId);
        return distance;
    }
// Next Location Search
public int localSearchForMovement(Layout selectedLoad){
    int nextLocation = 0;
    HashMap<Integer, Integer> neighboursAndDistance = new HashMap<Integer, Integer>();
    for(int i=0; i<getLoadNeighbours(selectedLoad).size(); i++){
        int distance = calculateDistanceBetweenLoads(getLoadNeighbours(selectedLoad).get(i),
                                                  map.get(selectedLoad.getLocation()).getDestination());
        neighboursAndDistance.put(getLoadNeighbours(selectedLoad).get(i), distance);
    }
    Map<Integer, Integer> neighboursByDistance = sortMapByValues(neighboursAndDistance);
    ArrayList<Integer> neighbourLocations = new ArrayList<Integer>(neighboursByDistance.keySet());
    ArrayList<Integer> neighbourLocationsByDistance = new ArrayList<Integer>(neighboursByDistance.values());
    ArrayList<Integer> feasibleNeighbourLocations = new ArrayList<Integer>();
    ArrayList<Integer> nextLocationOptions = new ArrayList<Integer>();
    for(int k = 0; k < neighbourLocations.size(); k++){
        if((neighbourLocationsByDistance.get(k) == Collections.min(neighbourLocationsByDistance))){
            nextLocationOptions.add(neighbourLocations.get(k));
        }
    }
    int i = 0;
    while(i < nextLocationOptions.size()){
        if(selectedLoad.getPriority() > map.get(nextLocationOptions.get(i)).getPriority() &&
           !map.get(nextLocationOptions.get(i)).getSelectionVariable() == 1 &&
           selectedLoad.getPriority() >= map.get(nextLocationOptions.get(i)).getInheritedPriority()){}
        feasibleNeighbourLocations.add(nextLocationOptions.get(i));
        i++;
    }
    TreeMap<Integer, Integer> feasibleNeighborsByPriorities = new TreeMap<Integer, Integer>();
    for(int k = 0; k < feasibleNeighbourLocations.size(); k++){
        feasibleNeighborsByPriorities.put(map.get(feasibleNeighbourLocations.get(k)).getPriority(), feasibleNeighbourLocations.get(k));
    }
    if(feasibleNeighbourLocations.size() > 0){
        nextLocation = feasibleNeighborsByPriorities.firstKey();
        map.get(nextLocation).setInheritedPriority(selectedLoad.getPriority());
        map.get(nextLocation).setSelectionVariable(1);
    } else {
        nextLocation = 0;
    }
    return nextLocation;
}

// Generating sequence of tagged modules
public ArrayList<Integer> sequenceOfTaggedModules(int currLocation, int nextLocation){
    ArrayList<Integer> sequenceOfTaggedModules = new ArrayList<Integer>();
    Set<Integer> unusedModulesList = new TreeSet<Integer>();
    sequenceOfTaggedModules.add(currLocation);
    while(map.get(nextLocation).getPriority() != 0){
        nextLocation = localSearchForEmptyLocation(sequenceOfTaggedModules.get(sequenceOfTaggedModules.size()-1), sequenceOfTaggedModules);
        if(nextLocation != 0){
            sequenceOfTaggedModules.add(nextLocation);
        }
    }
    return sequenceOfTaggedModules;
}
sequenceOfTaggedModules.add(nextLocation);
}
else{
    while(nextLocation ==0 && sequenceOfTaggedModules.size()>1)
    {
        nextLocation =
        localSearchForEmptyLocation(sequenceOfTaggedModules.get(sequenceOfTaggedModules.size()-2), sequenceOfTaggedModules);
        unusedModulesList.add(sequenceOfTaggedModules.get(sequenceOfTaggedModules.size()-1));
        sequenceOfTaggedModules.remove(sequenceOfTaggedModules.size()-1);
        if(nextLocation !=0){
            sequenceOfTaggedModules.add(nextLocation);
            break;
        }
    }
}
if(sequenceOfTaggedModules.size()<= 1){
    break;
}
}
if(sequenceOfTaggedModules.size()==1){
    resetLoadVariables(1,sequenceOfTaggedModules);
    sequenceOfTaggedModules.clear();
}
ArrayList<Integer> listOfModules = new ArrayList<Integer>(unusedModulesList);
resetLoadVariables(1,listOfModules);
return sequenceOfTaggedModules;
}
// Tagging modules for movement
public int localSearchForEmptyLocation(int currLocation, ArrayList<Integer> pathToEmptySpace){
    int nextLocation=0;
    TreeMap<Integer,Integer> neighboursAndPriorities = new TreeMap<Integer, Integer>();
    ArrayList<Integer> feasibleNeighbours = new ArrayList<Integer>();
    for(int i=0;i<getLoadNeighbours(map.get(currLocation)).size();i++){
        neighboursAndPriorities.put(map.get(getLoadNeighbours(map.get(currLocation)).get(i)).getPriority()
        ,getLoadNeighbours(map.get(currLocation)).get(i));
    }
    ArrayList<Integer> neighboursPriorities = new ArrayList<Integer>(neighboursAndPriorities.keySet());
    ArrayList<Integer> neighboursLocations = new ArrayList<Integer>(neighboursAndPriorities.values());
    int i =0;
    try{
        while(i <neighboursPriorities.size()){
            if(map.get(neighboursLocations.get(i)).getPriority() < map.get(currLocation).getInheritedPriority()
            && !(map.get(neighboursLocations.get(i)).getSelectionVariable()==1)
            && !pathToEmptySpace.contains(neighboursLocations.get(i))){
                feasibleNeighbours.add(neighboursLocations.get(i));
            }
            i++;
        }
    }
    TreeMap<Integer,Integer> feasibleNeighboursByPriorities = new TreeMap<Integer, Integer>();
    for(int k=0; k<feasibleNeighbours.size();k++){
        feasibleNeighboursByPriorities.put(map.get(feasibleNeighbours.get(k)).getPriority()
        , feasibleNeighbours.get(k));
    }
}
if (feasibleNeighbours.size() > 0) {
    nextLocation = feasibleNeighboursByPriorities.get(feasibleNeighboursByPriorities.firstKey());
    map.get(nextLocation).setInheritedPriority(map.get(currLocation).getInheritedPriority());
    map.get(nextLocation).setSelectionVariable(1);
} else {
    nextLocation = 0;
}
}

try{
    return nextLocation;
}

// Move Loads to empty location
public void moveLoadsToEmptySpace(ArrayList<Integer> pathToEmptySpace){
    int i = 0;
    while (i < pathToEmptySpace.size() - 1) {
        moveLoads(pathToEmptySpace.get(i), pathToEmptySpace.get(i + 1));
        if (orderList.contains(pathToEmptySpace.get(i))) {
            int index = orderList.indexOf(pathToEmptySpace.get(i));
            orderList.set(index, pathToEmptySpace.get(i + 1));
        } else if (orderList.contains(pathToEmptySpace.get(i + 1))) {
            int index = orderList.indexOf(pathToEmptySpace.get(i + 1));
            orderList.set(index, pathToEmptySpace.get(i));
        }
        i++;
    }
}

// Swap module attributes to indicate movement
public void moveLoads(int firstLocation, int secondLocation) {
    int priority1 = map.get(firstLocation).getPriority();
    int state1 = map.get(firstLocation).getState();
    int inheritedPriority1 = map.get(firstLocation).getInheritedPriority();
    Color color1 = map.get(firstLocation).getColor();
    int selectionVariable1 = map.get(firstLocation).getSelectionVariable();
    int orderVariable1 = map.get(firstLocation).getOrderVariable();
    int destination1 = map.get(firstLocation).getDestination();
    int orderListConflictVariable1 = map.get(firstLocation).getOrderListConflictVariable();
    int orderListTimeStepCount1 = map.get(firstLocation).getOrderListLoadTimeStepCount();
    int destinationTemp1 = map.get(firstLocation).getDestinationTemp();
    int destinationTempVariable1 = map.get(firstLocation).getDestinationTempVariable();
    ArrayList<Integer> taggedSequence1 = map.get(firstLocation).getSequenceOfTaggedModules();
    int priority2 = map.get(secondLocation).getPriority();
    int state2 = map.get(secondLocation).getState();
    int inheritedPriority2 = map.get(secondLocation).getInheritedPriority();
    Color color2 = map.get(secondLocation).getColor();
    int selectionVariable2 = map.get(secondLocation).getSelectionVariable();
    int orderVariable2 = map.get(secondLocation).getOrderVariable();
    int destination2 = map.get(secondLocation).getDestination();
    int orderListConflictVariable2 = map.get(secondLocation).getOrderListConflictVariable();
    int orderListTimeStepCount2 = map.get(secondLocation).getOrderListLoadTimeStepCount();
    int destinationTemp2 = map.get(secondLocation).getDestinationTemp();
int destinationTempVariable2 = map.get(secondLocation).getDestinationTempVariable();
ArrayList<Integer> taggedSequence2 = map.get(secondLocation).getSequenceOfTaggedModules();

//Swapping Values
map.get(firstLocation).setPriority(priority2);
map.get(firstLocation).setState(state2);
map.get(firstLocation).setInheritedPriority(inheritedPriority2);
map.get(firstLocation).setColor(color2);
map.get(firstLocation).setSelectionVariable(selectionVariable2);
map.get(firstLocation).setOrderVariable(orderVariable2);
map.get(firstLocation).setDestination(destination2);
map.get(firstLocation).setSequenceOfTaggedModules(taggedSequence2);
map.get(firstLocation).setOrderListConflictVariable(orderListConflictVariable2);
map.get(firstLocation).setOrderListLoadTimeStepCount(orderListTimeStepCount2);
map.get(firstLocation).setDestinationTemp(destinationTemp2);
map.get(firstLocation).setDestinationTempVariable(destinationTempVariable2);
map.get(secondLocation).setPriority(priority1);
map.get(secondLocation).setState(state1);
map.get(secondLocation).setInheritedPriority(inheritedPriority1);
map.get(secondLocation).setColor(color1);
map.get(secondLocation).setSelectionVariable(selectionVariable1);
map.get(secondLocation).setOrderVariable(orderVariable1);
map.get(secondLocation).setDestination(destination1);
map.get(secondLocation).setSequenceOfTaggedModules(taggedSequence1);
map.get(secondLocation).setOrderListConflictVariable(orderListConflictVariable1);
map.get(secondLocation).setOrderListLoadTimeStepCount(orderListTimeStepCount1);
map.get(secondLocation).setDestinationTemp(destinationTemp1);
map.get(secondLocation).setDestinationTempVariable(destinationTempVariable1);

} // Get Module Neighbors
public ArrayList<Integer> getLoadNeighbours(Layout selectedLoad){
    ArrayList<Integer> neighbourLocations = new ArrayList<Integer>();
    if(selectedLoad.getLeftLocation()!=0 ){
        neighbourLocations.add(selectedLoad.getLeftLocation());
    }
    if(selectedLoad.getRightLocation()!=0 ){
        neighbourLocations.add(selectedLoad.getRightLocation());
    }
    if(selectedLoad.getTopLocation()!=0 ){
        neighbourLocations.add(selectedLoad.getTopLocation());
    }
    if(selectedLoad.getBottomLocation()!=0 ){
        neighbourLocations.add(selectedLoad.getBottomLocation());
    }
    return neighbourLocations;
}

// Method for sorting maps(data structures)
@SuppressWarnings("hiding")
public static <Integer, Object extends Comparable<? super Integer>> Map<Integer, Integer> sortMapByValues(Map<Integer, Integer> map )
{
    List<Map.Entry<Integer, Integer>> list =
        new LinkedList<Map.Entry<Integer, Integer>>( map.entrySet() );
    Collections.sort( list, new Comparator<Map.Entry<Integer, Integer>>()
    {
```java
@SuppressWarnings("unchecked")
public int compare( Map.Entry<Integer, Integer> o1, Map.Entry<Integer, Integer> o2 )
{
    return ((Comparable<Integer>) o1.getValue()).compareTo( o2.getValue() );
}

Map<Integer, Integer> result = new LinkedHashMap<Integer, Integer>();
for (Map.Entry<Integer, Integer> entry : list)
{
    result.put( entry.getKey(), entry.getValue() );
}
return result;

// Display load movement
public void displayMap(int sleepTime){
    try {
        Thread.currentThread();
        Thread.sleep(sleepTime); // PrintStackTrace();
        warehouseMap.remove(WarehouseMap.panel);
        WarehouseMap.Warehouse2DPanel panel = new WarehouseMap.Warehouse2DPanel();
        warehouseMap.add(panel);
        warehouseMap.validate();
        warehouseMap.setVisible(true);
    }
    catch (InterruptedException e){
        e.printStackTrace();
    }
}

/* Generating random colors for packages*/
package priorityBasedMovement;
import java.awt.Color;
import java.util.Random;
public class RandomColorGenerator
{
    private Random rand;
    public RandomColorGenerator()
    {
        rand = new Random();
    }
    public Color randomColor()
    {
        Color randomColor =new Color(rand.nextInt(256),rand.nextInt(256),rand.nextInt(256));
        int flag =0;
        while(flag ==0){
            if(randomColor != Color.white || randomColor != Color.orange
                || randomColor != Color.black){
                flag =1;
            }else{
                randomColor = new Color(rand.nextInt(256),rand.nextInt(256),rand.nextInt(256));
                flag=0;
            }
        }
    }

```
package priorityBasedMovement;
import java.awt. *;
import java.awt.Color;
import javax.swing. *;
import javax.swing.border. *;
import java.awt.event. *;
import java.util. *;
@ SuppressWarnings("serial")
public class WarehouseMap extends JFrame{
    static Warehouse2DPanel panel = new Warehouse2DPanel();
    public WarehouseMap(){
        setTitle("Public Distribution Center - Master's Thesis Research - Mohit A Datar ");
        getContentPane().setLayout(new BorderLayout() );
        setSize(750, 750);
        getContentPane().add(panel);
        setDefaultCloseOperation( EXIT_ON_CLOSE );
    }
    public static class Warehouse2DPanel extends JPanel{
        Warehouse2DPanel(){
            Layout[][] layout = Layout.layout;
            setLayout(new GridLayout(layout.length, layout[0].length));
            JPanel [][]wall = new JPanel[layout.length][layout[0].length];
            for(int i=0; i<layout.length; i++){
                for(int j=0; j<layout[0].length; j++){
                    wall[i][j] = new JPanel();
                    if(layout[i][j].getOrderVariable()==1){
                        String priority = "\n\n"+String.valueOf(layout[i][j].getPriority());
                        JLabel labelPriority = new JLabel(priority,JLabel.CENTER);
                        labelPriority.setForeGround(Color.white);
                        wall[i][j].setBackGround(layout[i][j].getColor());
                        wall[i][j].setBorder(BorderFactory.createMatteBorder(3,3,3,3, Color.black));
                        wall[i][j].add(labelPriority);
                    }
                    else if(layout[i][j].getState()==0 && layout[i][j].getPriority()==0){
                        wall[i][j].setBackGround(layout[i][j].getColor());
                        wall[i][j].setBorder(BorderFactory.createLineBorder(Color.black));
                    }
                    else{
                        String priority = "\n\n"+String.valueOf(layout[i][j].getPriority());
                        JLabel labelPriority = new JLabel(priority,JLabel.CENTER);
                        labelPriority.setForeGround(Color.white);
                        wall[i][j].setBackGround(layout[i][j].getColor());
                        wall[i][j].setBorder(BorderFactory.createLineBorder(Color.black));
                        wall[i][j].add(labelPriority);
                    }
                }
                if(Layout.docks.contains(layout[i][j].getLocation())){
                    String state = "\nDock";
                    JLabel labelState = new JLabel(state,JLabel.CENTER);
                    wall[i][j].setBackGround(layout[i][j].getColor());
                    wall[i][j].setBorder(BorderFactory.createLineBorder(Color.black));
                    wall[i][j].setToolTipText("Location id: "+layout[i][j].getLocation());
                }
            }
        }
    }
}
wall[i][j].add(labelState);
} add(wall[i][j]);
} }